[redacted]

UIUCDCS-R-73-551

math

COO-2118-0041

ILLIAC III COMPUTER SYSTEM MANUAL: ARITHMETIC UNITS--VOLUME 2

by

Lakshmi N. Goyal

January 1973

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-73-551

ILLIAC III COMPUTER SYSTEM MANUAL:

ARITHMETIC UNITS*

VOLUME 2

by

LAKSHMI N. GOYAL

January 1973

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

* This work was supported by the U.S. Atomic Energy Commission under Contract No. AT(11-1) 2118.

ACKNOWLEDGMENT

The present work is concerned with the Control Logic and the hardware implementation of the arithmetic algorithms of Illiac III Arithmetic Units. This volume has been made possible by the contributions of many people. The author would like to express his appreciation to these people in general. In particular, the author will like to mention a few principal names. Dr. Daniel E. Atkins and Prof. James E. Robertson provided the original conceptual design. Dr. Atkins was mainly responsible for the design of the processing hardware and the arithmetic algorithms. These arithmetic algorithms are the basis for the control sequences reported in this work. The author is very grateful to his friend, Dr. Atkins, for his gracious help and time which he so ungrudingly gave, in the clarifications and understanding of the arithmetic algorithms. Mention should be made of Mrs. Tuh-Kai Koo who wrote extensive simulation programs which were used in validating the arithmetic algorithms. The author would also like to thank Professor Bruce H. McCormick under whose overall direction this work was undertaken and who taught the author the merits of logical segmentation of control logic in terms of Procedures and Subroutines.

The logic design and different versions of Control Points used in designing the control logic evolved from the contributions of many people. The principal contributors were Dr. Daniel E. Atkins, Dr. J. Divilbiss, Dr. B. J. Nordmann, Mr. S. Paul Krabbe, Mr. Ron Martin, Mr. Val Tareski and the author.

The author is very thankful to Mr. S. Paul Krabbe under whose direct supervision this control logic has been layed out on logic cards and their fabrication completed. He has borne the weight of the ardous task of supervising the wiring tables generation, back panel wiring, logic cards check out, etc.

In addition, he is chiefly responsible for the design of interface driver logic and numerous other hardware and circuit details.

The illustrations were prepared by Mr. Stan Zundo who labored over many revisions of the illustrations and drawings. His humor and cheerful disposition always made it a pleasure to go to the drafting section.

Thanks are to due to Mrs. Barbara Bunting and Mrs. Barbara Armstrong for doing a good job of typing and finally to Mr. Dennis Reed for the meticulous reproduction of this manual.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

This report is the second volume of Illiac III computer system manual on arithmetic units. The first volume described the various arithmetic orders to be executed in the arithmetic unit and the internal static description or in other words, the processing hardware of the Arithmetic Unit. This volume describes the control logic hardware and gives its operational description.

This report assumes that the reader is familiar with previously published reports and documents regarding the arithmetic unit. In particular the familiarity with the following documents is very essential.

DCS Report No. 366: "Illiac IV Computer System Manual: Arithmetic Units, Volume 1 by D. E. Atkins - This describes the processing hardware consisting of various registers, data paths, SDS-array, multiplier, recoder, model division and shifting network of the arithmetic unit.

DCS Report No. 418: "Arithmetic Unit of Illiac III: Simulation and Logical Design - Part II by P. L. Koo, D. E. Atkins. Revised by L. N. GOYAL - November 10, 1970". - This report describes the control algorithms for the execution of various arithmetic orders and their simulation at a very detailed, almost hardware level.

The present report does not describe the control algorithms in detail but assumes that the reader is familiar with them from Report No. 418. The author recommends that the present report should be read after reading once more the Report No. 418 and then keeping Report No. 418 always in close proximity.

This report is divided into various sections and subsections. The major sections are Strategy of Control Design, Control Point Flow charts and their description, Control Logic and Processing Hardware Interface and functional description of control signals.

Section 2 describes the technique of control point design, the various variations of control points, conventions and notations both in control point flow charts and the various logic drawings that implement them and can be read independently of any other section.

Section 3 gives the control point charts for the control sequences and subsequences necessary for the various arithmetic orders to be executed in the arithmetic unit. An operational description of the control sequences and subsequences is also given, both in an overall global sense as well as in more detailed local sense. Besides the control sequences for various arithmetic orders, some other related logic is described and is grouped under the heading "Miscellaneous".

Section 4 describes briefly the interface logic between the control logic hardware and the processing hardware which have been implemented with different logic families - the former with TTL and the latter with DTL.

Section 5 shows an index of the control logic drawings, followed by a set of these drawings.

Section 6 is an alphabetized list of control signal names together with their functional definition. This list includes many signal names whose definition already appears in Volume 1, DCS Report No. 366.

Finally, there is an appendix at the end.

It should be noted that this report is a working document for the design, construction, checkout and maintenance of the arithmetic unit for Illiac III. It is an evolving document and will be updated specially as the checkout reveals any deficiencies. Readers are encouraged to bring to the author's attention any errors and suggestions.

2. STRATEGY OF CONTROL DESIGN

2.1 General Remarks

The control of the arithmetic units essentially consists in executing a time-ordered sequence of elementary micro-operations like operat-ating of gates, the setting or clearing of a flip-flop etc. This time-ordered sequence can best be illustrated as in computer programming, with a flow chart. Each control sequence in the arithmetic unit (AU) has been broken down into control steps. A control step consists of two stages: the Task Stage and the Sequence Stage. In the task stage, elementary micro-operation(s) is(are) performed conditional upon status conditions on the processing hardware i.e., gates are operated, counters incremented etc. Within the sequence stage, the decision is made as to which task stages to initiate next. The sequence stages are interleaved with the task stages. The control steps are performed by logical circuits called control points. Hence the control flow charts are called Control Point Flow Charts.

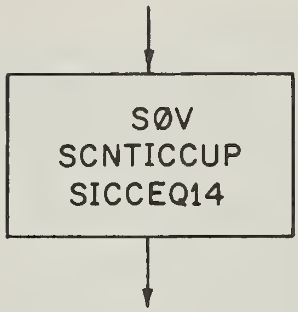
2.2 Control Point Flow Chart Conventions

A control point flow chart essentially consists of task stages interleaved with sequence stages besides the entry point(s) to and exit point(s) from the flow chart.

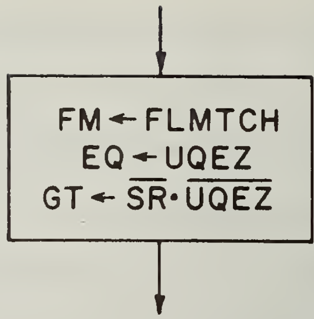
2.2.1 Task Stage:

A task stage indicates all the elementary micro-operations, conditional and/or unconditional, that can be performed at that instant of time in the time-ordered sequence of events. This elementary micro-operation or a set of these micro-operations (unconditional or conditional on the same status condition) is called a task. A task is designated by a rectangular box with an input from the top and an output, a reply, from the bottom such as shown in Figure 2.2.1.1a. The arrowheads indicating directions of flow are optional. If they are not shown, flow is assumed to be from top to bottom. Within the task box are written the signal names which are operated to execute the task, when the task box is entered during the control flow. However, at times, instead of the signal names, we write a micro-operation in the format $X \leftarrow Y$ where X is the dependent variable (a flip-flop in actual implementation) and Y is the independent variable. This is shown in Figure 2.2.1.1.b. Associated with each task is a certain duration of time after which a reply signal is generated and control proceeds along the exit line of the box.

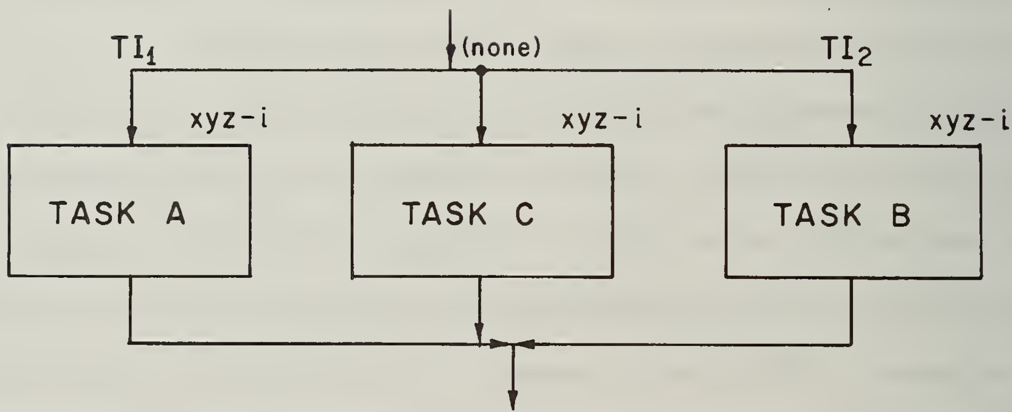
When more than one task is initiated concurrently, the conditions which must exist for the task to take place are indicated above the task entry line as shown in Figure 2.2.1.1c. The task condition "none" indicates that the initiation of task requires only an initiating signal from the previous stage



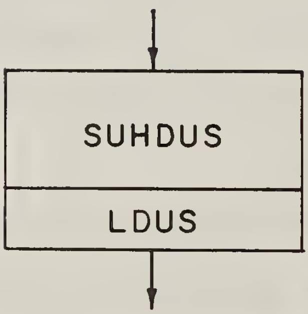
(a) Task Box



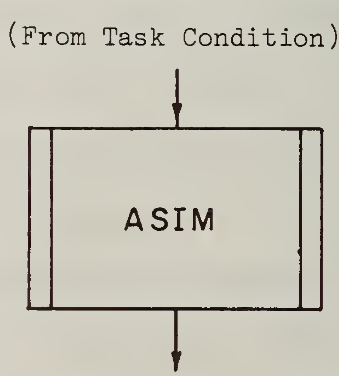
(b) Task Box



(c) Task Stage



(d) Delayed Task Generation (DETAG)



(e) Subroutine (Control Subsequence) Call

Figure 2.2.1.1. Symbols Used in a Task Stage

and will always be performed once that task stage is entered. Note that the task conditions need not be mutually exclusive. The tasks whose reply signals merge on the same horizontal line are assumed to be of the same duration. In terms of hardware, this means that they may share the same timing model and hence all these task signals are capable of being generated by a single control point. The delay of the timing model used in the corresponding control point is $\geqslant$ the largest operating time of all the tasks whose reply lines merge on the same horizontal line.

Another symbol used, at times, in the task stages of the control flow chart is a rectangular task box divided by a horizontal line towards the bottom as depicted in Figure 2.2.1.1.d. The task signals are written in the box both above and below the line. The interpretation of this box is that the signals written below the line are to be turned on, a controllable delay later than the task signals written above the line. Such task boxes are very suitable for indicating the loading of a register from another. The signal above the line will select the source register to the input of the register being loaded. After the selector gets settled, the load signal (essentially equivalent to a gating signal) written below the line is turned on. In this way one can affect a savings in control points during the implementation.

Still another rectangle with a dividing vertical line at each end of the box (Figure 2.2.1.1.e) is used for indicating a subroutine or procedure call. This symbol is identical to the predefined process symbol of ANSI flow chart standards. The control sequence of the arithmetic units has been divided into Procedures and Subroutines. A subroutine is a control logic subsequence which can be called from different parts of the same sequence or even from different

control sequences. The obvious reason for this type of structure is that it reduces the logic by allowing common subsequences to be done using the same physical logic. Similarly a Procedure is also a control logic subsequence. However, the difference between a Subroutine and a Procedure is that when a control subsequence corresponding to the Subroutine is finished, the reply signal from the subsequence must return to the point, where it was called in the main sequence so that the control flow can continue. But, in the case of the Procedure, the control flow essentially branches off from the main sequence, and at the end of the Procedure subsequence, the reply signal can continue to do other tasks, call Subroutines or go to other Procedures. The advantage of so dividing the control logic sequence into Procedures and Subroutines is that it is easier to visualize conceptually when it is placed in such a modular format, and is also easier to debug and checkout. In actual hardware, a subroutine call is implemented by a special kind of Control Point called Calling Control Point. (For more details see section 2.3.1.3). In the symbol for subroutine call task, is written the name of the subroutine. The task signal initiates the subroutine control logic subsequence, and the reply from the subsequence is returned to the task stage where this subsequence was called (i.e., initiating signal was generated) so that the control flow for the sequence can continue. It may be noted, however, that, at times, the names of the other task signals may also be written in the box in addition to the subroutine name. This means that these elementary micro-operations are to be done in parallel with the execution of the subsequence.

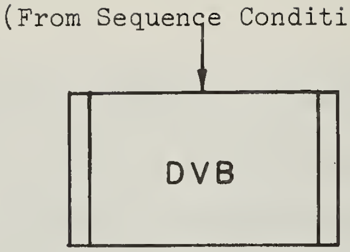
A set of these task blocks, external conditions for entry to a task box (e.g. contents of a register equal zero) and reply generation facilities constitute a Task Stage. Each task rectangle is labelled with the name of that task stage. The label has the format XYZ-i where XYZ is the name abbreviated from the sequence name and i is a numerical number. This label is also the name of the control point executing this task stage.

2.2.2 Sequence Stage

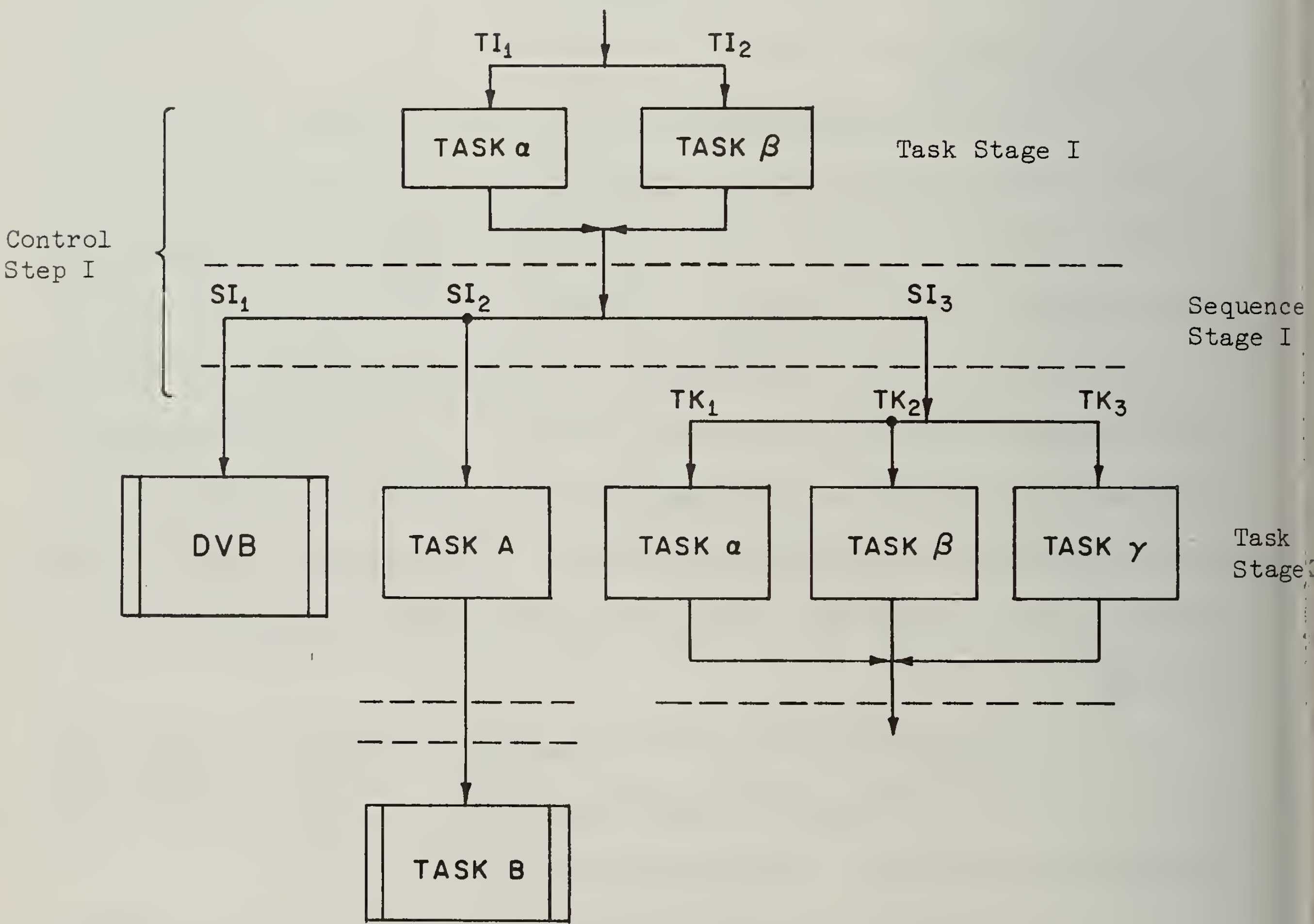
Interleaved between task stages are sequence stages: that portion of control which determines which task step (stage) or subsequence is executed next. The sequence may cause a branch to another subsequence of control flow. This subsequence called a Procedure is identified by the same symbolic box as for Subroutine call in the task stage, with the name of the Procedure subsequence written inside the rectangle as shown in Figure 2.2.2.1.a. This facilitates the description of the Procedure Subsequence on a different flow chart of the same main sequence. The branching conditions or the Sequence Conditions (SIj) are shown in the same way that the Task Conditions (TIj) are shown in the task stage.

A task stage followed by a sequence stage constitutes a Control Step. On the flow charts, the boundary between task stage and sequence stage is indicated by horizontal dashed line. It is shown in Figure 2.2.2.1.b.

In the sequence and task conditions SIj and TIj respectively, I stands for the number of the control step which is the same as numerical number i in the name of the task stage and j identifies the number of the condition (task



(a) Sequence Stage Branch to a Procedure (Control Subsequence)



(b) A Task Stage Followed by a Sequence Stage

Figure 2.2.2.1. Sequence Stage and Control Step Illustration

or sequence) within that control step. The Boolean equations for the task and sequence conditions are also shown on the flow chart.

Note that the Procedure subsequences (or branches) are initiated by the sequence steps and Subroutine subsequences are initiated by task steps.

2.2.3 "Entry" and "Exit" Symbols

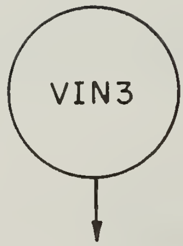
The single circle(s) represent(s) the Entry point(s) to a new control block (i.e., a subsequence) or a new page of the flow chart when it appears at the top of the page. The double circle indicates the Exit from the page, when the whole sequence or subsequence can't be fitted on one page. These two symbols are shown in Figures 2.2.3.1.a and 2.2.3.1.b respectively.

In case of multiple entry points to a flow chart, the name of the control step where the entry takes place is written in the Entry circle. In case of single entry, where no confusion is created, the name of the control step may or may not be written.

Every double Exit Circle must have a corresponding entry circle in the collection of flow charts. Inside the Exit Circle, we write the same name as the name in the corresponding entry circle.

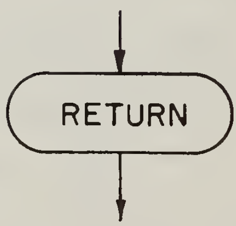
Note that an exit from a page or control flow also takes place when the sequence stage causes a branch to a Procedure subsequence indicated by the Procedure/Subroutine rectangle.

Finally, a terminal symbol (Figure 2.2.3.1.c) also appears on the flow charts which indicates the end of the main sequence or a Subroutine subsequence. In case of Subroutine, the word 'RETURN' is written inside the terminal symbol signifying that the control should return to the calling point in the



(a) 'Entry' Symbol

(b) 'Exit' Symbol



(c) Subroutine Terminal Symbol

Figure 2.2.3.1. Control Point Flow Chart "Entry", "Exit" and Subroutine Terminal Symbols

main sequence. In case of the end of the sequence, the word 'OUT' is written to indicate that the sequence is ended and the control should go out of this sequence.

Figures 3.7.2.1 and 3.7.2.2.1 illustrate an example of use of task and sequence stages, entry and exit circles, etc. in one control point flow chart.

2.2.4 "JOIN" Symbol

This is required when two control subsequences are initiated in parallel and if the control flow demands that both the subsequences be finished before the control flow can proceed further. A symbol shown in Figure 2.2.4.1 and resembling a Pentagon called "JOIN" is used to indicate that the replies from both the subsequences must be true before the control flow can go on. An example of its use is given in the flow chart for GETDEC in Figure 3.8.4.2.2.1.

2.2.5 Miscellaneous Notations

It was stated earlier that in the Delayed Task Generation (DETAG) rectangle, the task signal below the horizontal dividing line was the delayed task signal. In some cases, no task signal name appears below the horizontal dividing line, but rather a wait is indicated for some logical circuit or condition to settle down. This notation is used simply to indicate that a Control Point with two cascaded timing models are used in the physical realization of Control sequence. For an example, see the DETAG rectangle CVD-11 in the flow chart for GETDEC in Figure 3.8.4.2.2.1.

Sometimes, instead of task signal names, the word "Dummy" is written inside the task box. It indicates that no control signals are activated and

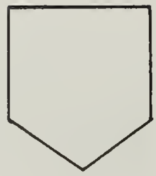


Figure 2.2.4.1. "JOIN" Symbol for Interlocking Parallel Control Chains

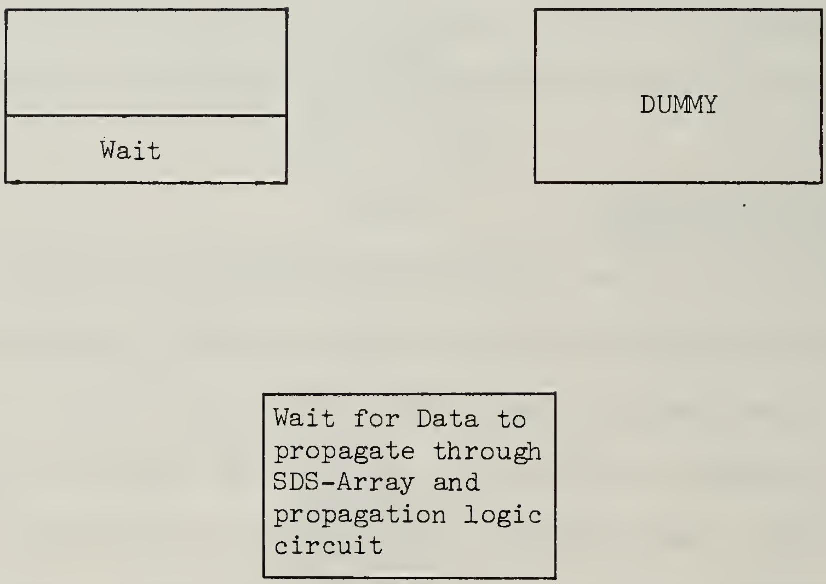
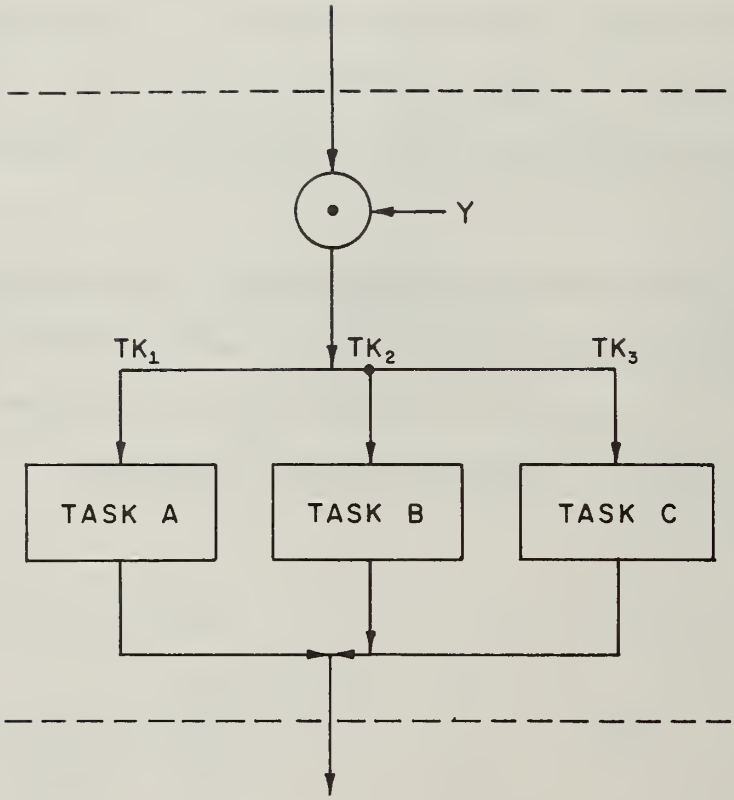


Figure 2.2.5.1. Miscellaneous Symbols Used in a Task Stage

has no logical significance, but is added for hardware considerations. Similarly, instead of the word "Dummy", a wait may be indicated for some logical condition or circuit to stabilize. An example of each of these occurs in the control-point flow charts for DVB and CAL respectively as shown in Figures 3.8.2.2.3.2 and 3.5.2.2.

When conditions other than Manual ENable (MEN) are present on the ENABLE (EN) input of a control point, these conditions are indicated as shown in Figure 2.2.5.2. If no conditions are shown, then it is assumed that the control point is enabled unless the maintenance stop is activated.

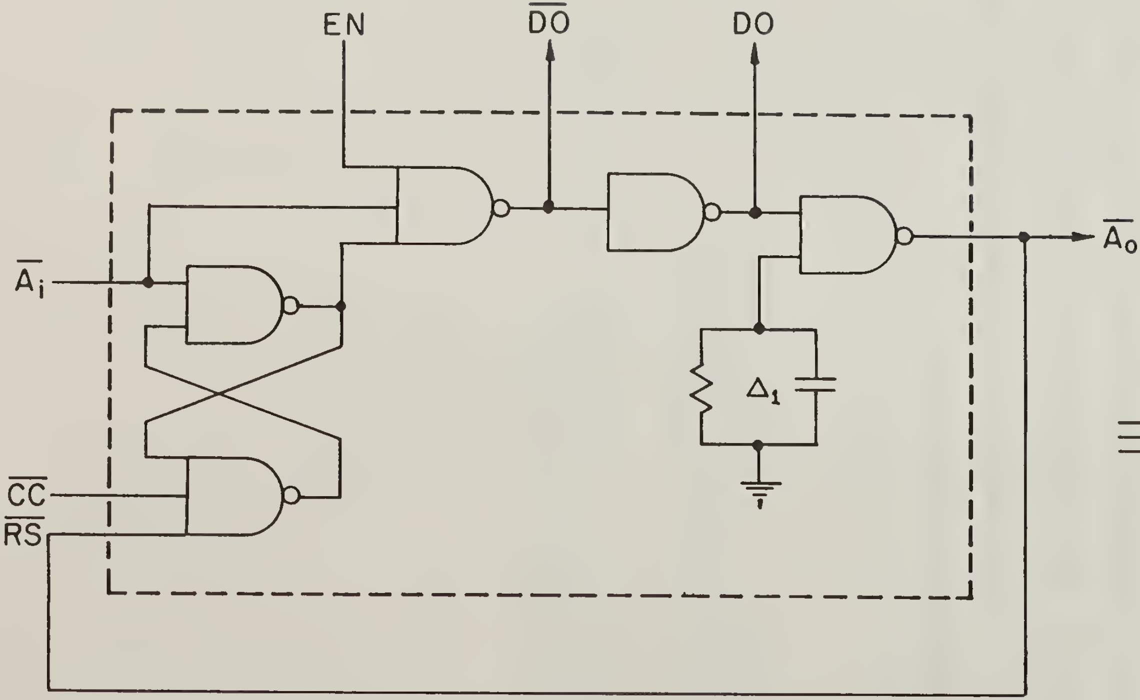


Y = Boolean expression to represent
the logical condition on EN input
of corresponding control point

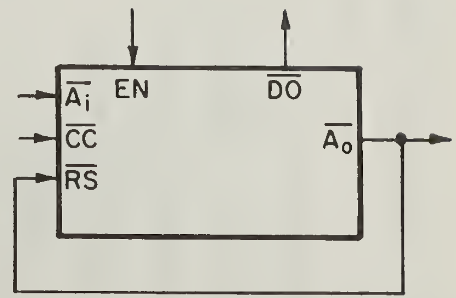
Figure 2.2.5.2. Representation of a Logical Condition on the 'Enable' Input of a Control Point for Asynchronous Operation

2.3 Logic Implementation of Control-Point Flow Charts

As stated in the previous section, the control flow charts specifying the time-ordered sequence of micro-operations are implemented in logic by a basic building block called control point (Figure 2.3.1) and its few variants. The flow charts are specifically so drawn that there is one to one correspondence between the flow charts and the logic implementation. The task stage of each control step is implemented by the control point and the sequence stage is realized by a sequence stage combinational logic. The basic control point used in the Arithmetic Units' Control is the same as described by R. Martin in DCS Report No. 400. For completeness sake, his description is reproduced in the appendix at the end. However, certain modifications have been made, and they are described in the following sections.



Detailed Logical Diagram



Equivalent Symbol

Figure 2.3.1. Martin's Basic Control Point

2.3.1 Modifications to Martin's Control Point

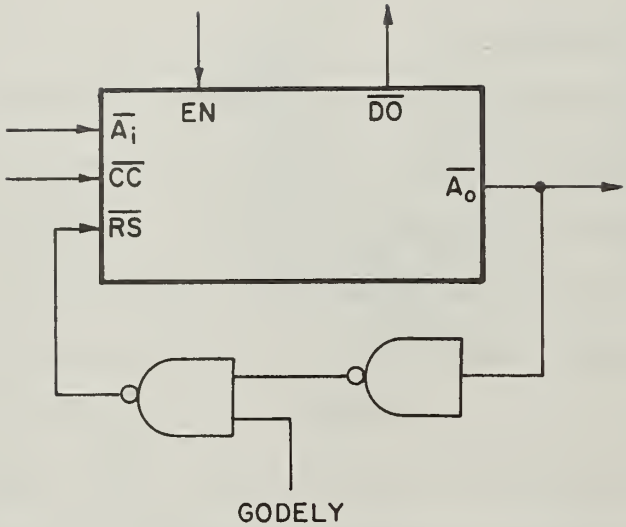
2.3.1.1 Memory Element Reset Logic

A modification has been made in the feedback path between Advance-out, $\overline{\text{Ao}}$, and the reset input, $\overline{\text{Rs}}$, of the memory element in the control point. A signal called $\overline{\text{GODELY}}$ is used to delay the resetting of the memory element of the presently active control point which in turn delays the priming and/or initiation of the next control point in the control flow sequence, as shown in Figures 2.3.1.2/1. This facility of $\overline{\text{GODELY}}$ is used for single stepping through the control points during the check-out. Note that in case of Martin's strategy, configuration of Figure A.2.2 using EN gate was used for diagnostic and check-out purposes.

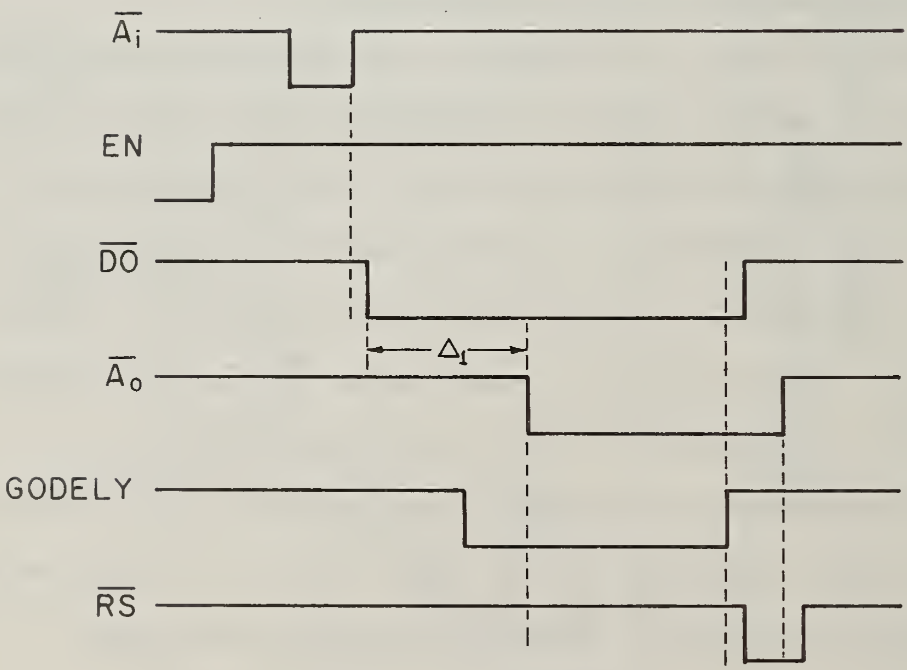
2.3.1.2 Delayed Task Generation (DETAG) Control Point

As mentioned earlier in Section 2.2.1, the task stage uses a DETAG rectangle in which the control signals written below the horizontal dividing line should be turned on a little delay later than those written above the line in the box. This is physically realized by essentially having two cascaded (in series) timing models. However, the other timing model is placed in the feedback path and the regular $\overline{\text{Ao}}$ becomes second $\overline{\text{DO2}}$ (Do Task) signal. The Advance-out, $\overline{\text{Ao}}$', is produced as shown in the Figure 2.3.1.2.1.

The same configuration of cascaded timing models is also used when the delay needed in the timing model is very large. This is to avoid the use of a large capacitor at one input of the Nand gate of the timing model.

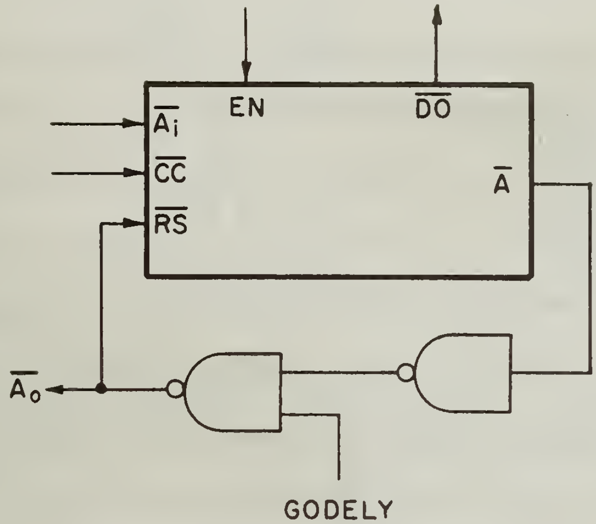


(a) Symbolic Representation

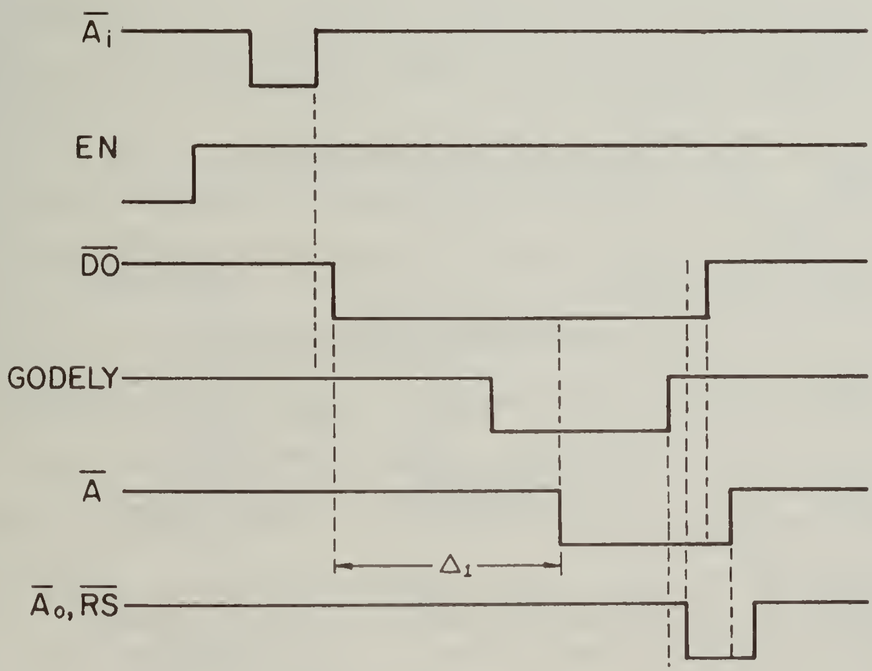


(b) Timing Diagram

Figure 2.3.1.1. Martin's Control Point with a Modified Memory Element Reset-Logic with <u>Delayed Initiation Only</u>

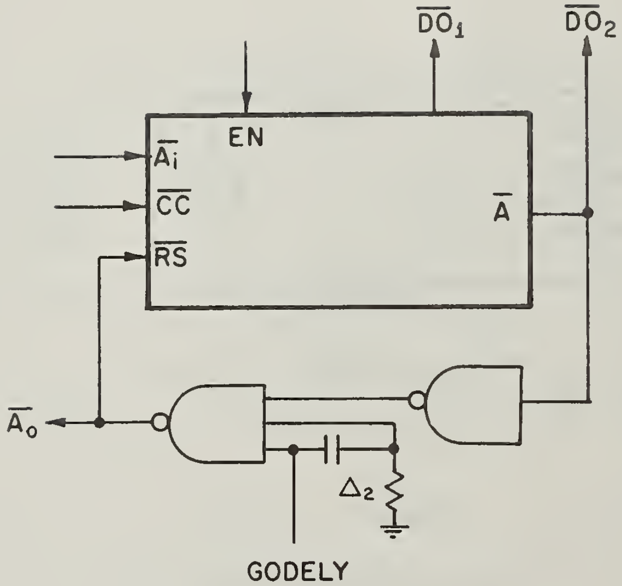


(a) Symbolic Representation

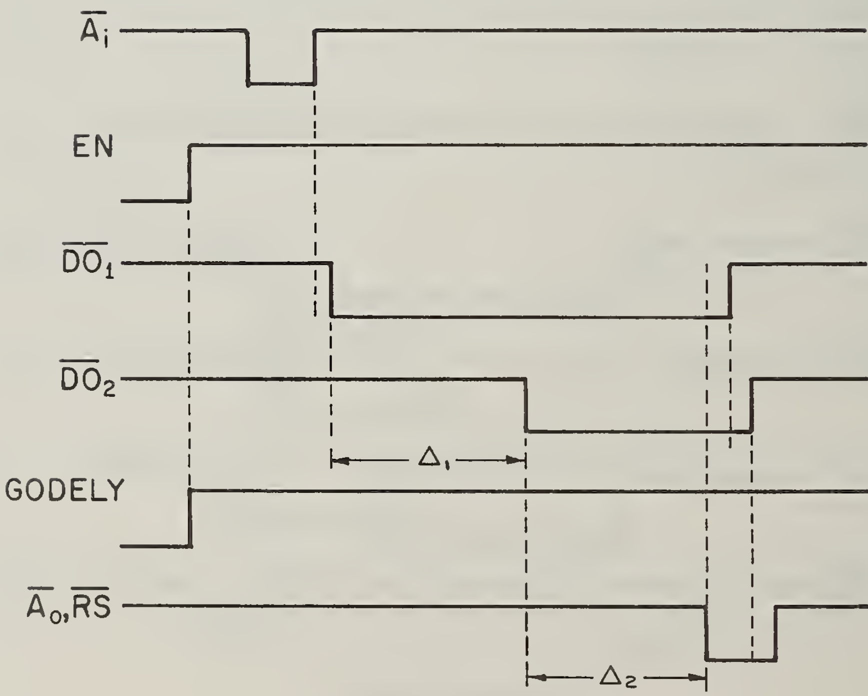


(b) Timing Diagram

Figure 2.3.1.2. Martin's Control Point with a Modified Memory Element Reset Logic with Delayed Priming and Initiation of Next Control Point



(a) Symbolic Representation



(b) Timing Diagram

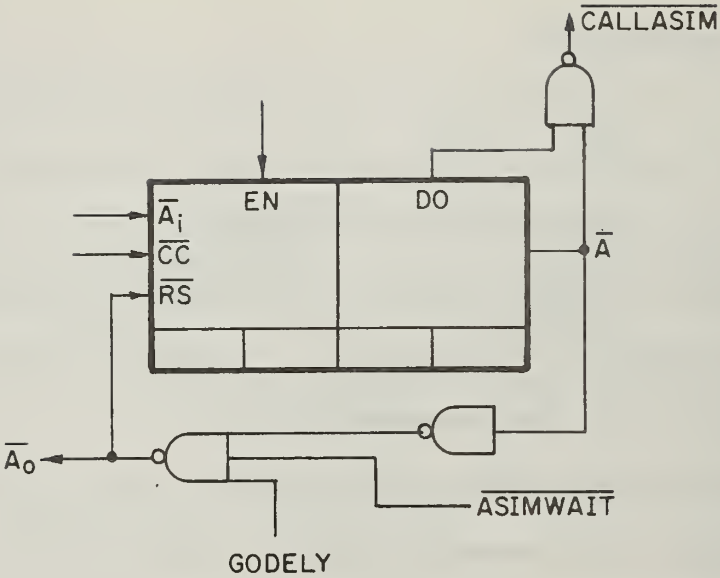
Figure 2.3.1.2.1. Delayed Task Generation Control-Point, its Symbolic Representation and Timing Diagram

2.3.1.3 Calling Control Point

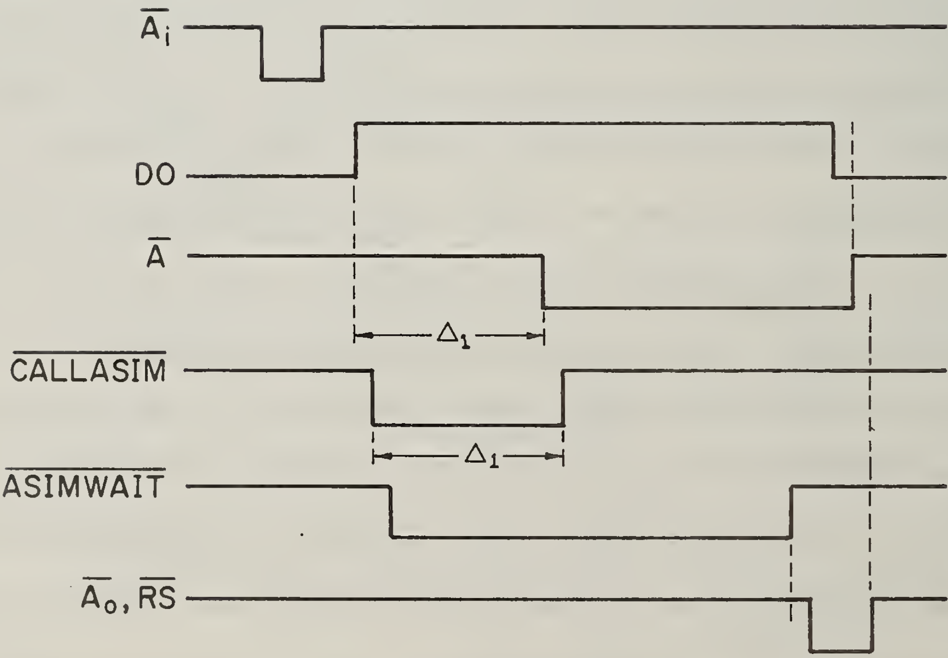
As mentioned in Section 2.2.1, the calling control point is used by a control sequence to call another subsequence just as the main program calls a subroutine. The logic implementation and operation of calling control point is exactly indentical to Martin's calling control point as described in Section A.2 of the appendix. Figure 2.3.1.3.1 shows the circuit configuration and timing diagram for the calling control point.

The calling control point (CCP) generates a pulsed task signal which has a labelling format 'CALL Subroutine Name'. It goes to set a control flip-flop and at the same time initiate the first control point of the called subroutine control sequence. This control flip-flop appears on the same logic drawing and the same logic card as the subroutine control sequence. As soon as this control flip-flop is set by the calling control signal, a reply signal from the quiescently '1' side of the flip-flop output goes down and inhibits the resetting of calling control point. As soon as the called subroutine control sequence has finished, the control flip-flop is reset and the reply signal goes up to '1' state to remove the inhibition on control point reset logic. If no other control signal, for example, GODELY, wants to inhibit the resetting of control point, the calling control point is reset and at the same time, the resetting control signal primes the next control point in sequence.

For an example of the calling Control Point and its use, see the control subsequence MPYEND whose control point flow chart is shown in Figure 3.6.2.2 of Section 3.6.2. In this subsequence, task stage MPY8 is implemented with a calling control point MPY8 shown in Drawing AUO-07-361-02 and calls the subroutine control sequence ASIM. Logic implementation of control subsequence ASIM is



(a) Symbolic Representation



(b) Timing Diagram

Figure 2.3.1.3.1. Calling Control-Point, its Symbolic Representation and Timing Diagram

shown in Drawing AUO-07-371-04. The control signal $\overline{\text{ASIMWAIT}}$ is the reply signal which inhibits the resetting of calling control point MPY8 till the resetting signal of control point ASM2 resets the control flip-flop.

It should be clearly noted that since the logic cards for the called control subsequence may be located quite far from the logic of calling control sequence, the time dealy in calling control point (which also determines the width of pulsed calling task signal) should be large enough so that the reset inhibit reply control signal from the subroutine wait-reply control flip-flop becomes effective lest the calling control point should be reset prematurely.

2.3.1.4 Interlocking of Two Parallel Independent Control Chains

Whenever two (or more) independent control chains (subsequences) are started in parallel, a flip-flop for each control chain is set to indicate that the control chains are active. When each control subsequence finished (in general they will finish at different times), each resets its corresponding active flip-flop. When both (or all in case of more than two) flip-flops are reset, then and only then the control is allowed to advance to the next control point in sequence. This approach, though slightly expensive because of the use of an extra flip-flop for each subsequence, is more useful for diagnostic purposes, since the active status of the subsequence is indicated by the true output of the flip-flops. This is shown in the Figure 2.3.1.4.1.

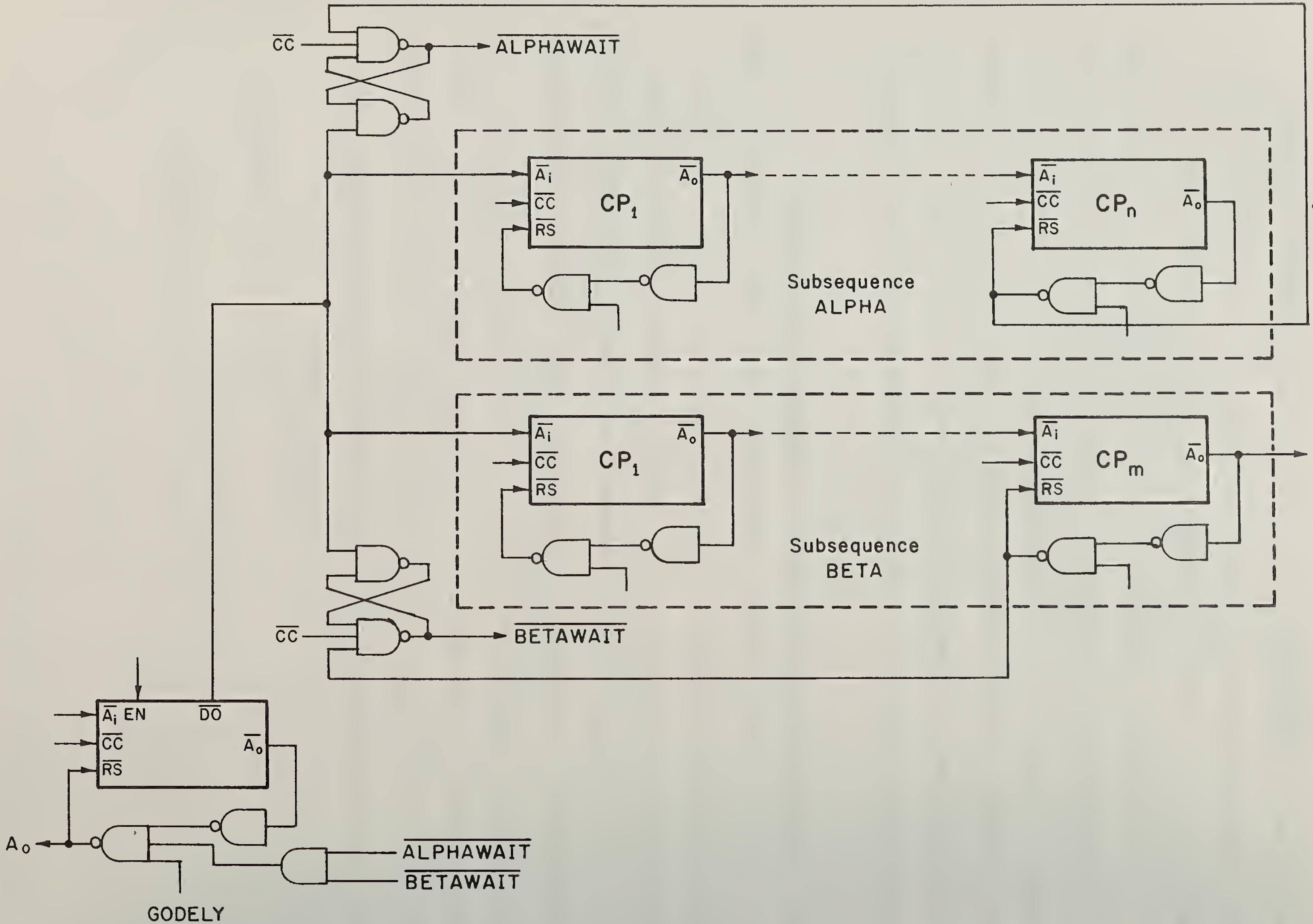


Figure 2.3.1.4.1. Interlocking of Two Parallel Independent Control Chains (Subsequences)

2.3.2 Conversion of Control Flow Charts into Control Logic Drawings

Logic implementation of the flow charts result in three broad categories of drawings. They are:

a) Control Point Drawings (CPD),

b) Task Signal Collector Drawing (TSCD) and

c) Task Driver Drawing (TDD).

2.3.2.1 Control Point Drawing

CPD's are the drawings which implement the task and sequence stages of the flow charts with the help of control points and combinational logic. The control flow charts are so designed and drawn that there is almost one to one correspondence between the logic drawing and the control flow charts. The task stage is implemented by an appropriate control point and the sequence stage by a combinational logic. As explained earlier, there are three kinds of 'rectangular' symbols used in the task stages of Figure 2.3.2.1.1. shows the corresponding control point symbolic configuration used to implement the task stage. (See also Section 2.3.3.1.)

In most cases, each flow chart has one separate logic drawing, although in a few cases, one flow chart has its corresponding logic implementation spread out on more than one drawing. Similarly, two flow charts may share the same logic drawing. Each logic drawing is given a title which is the same as the name (or names) of the Control Sequence, Procedure, or Subroutine, the logic drawing is implementing. This helps in correlating the flow charts and the logic drawings. Further, the names of the control point implementing a given task stage is the same as that of the task stage (i.e. the

Control Point Flow Chart Task Stage Symbol ≡ Corresponding Logic Drawing Control Point Symbol

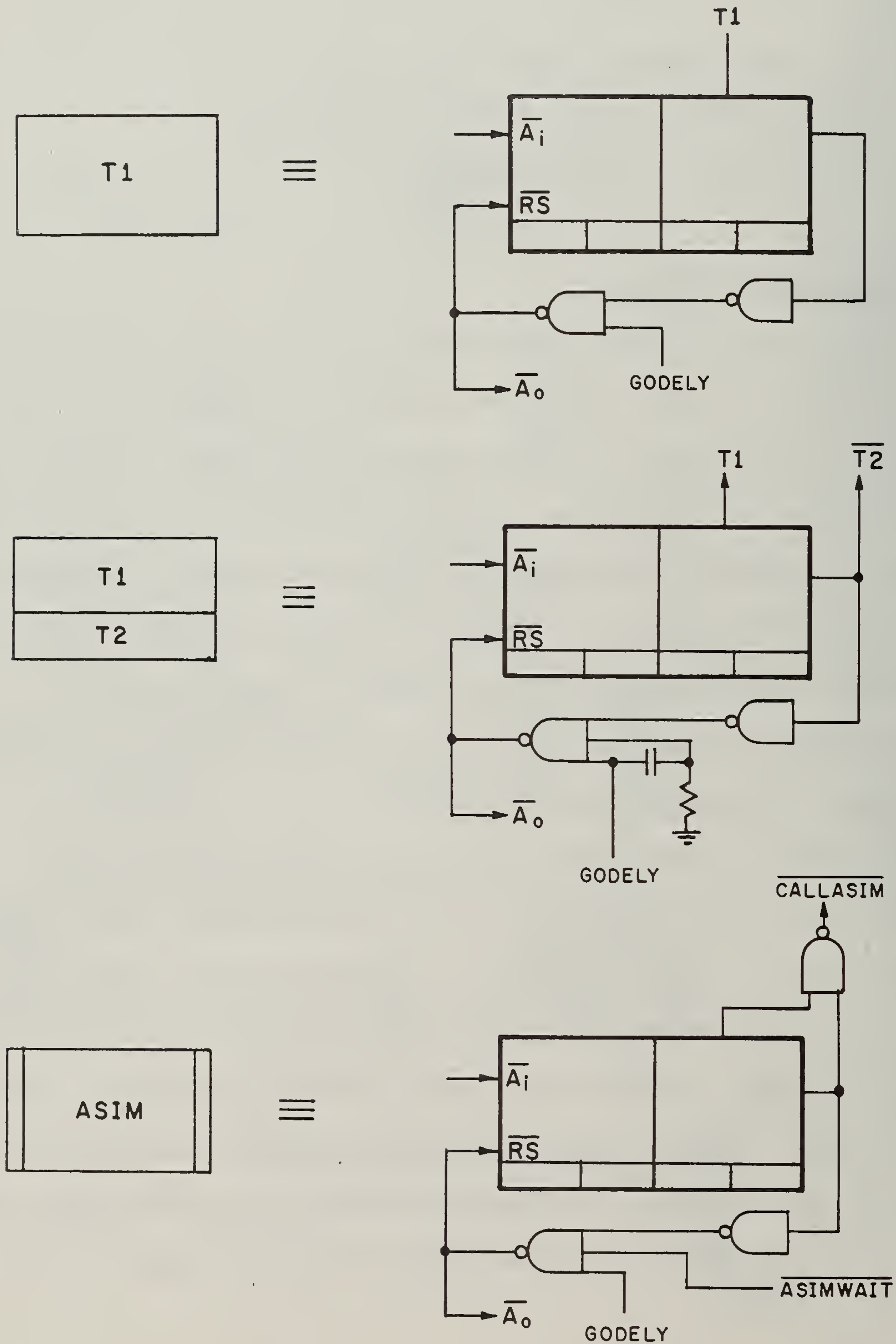


Figure 2.3.2.1.1. Task Stage Symbols and Corresponding Control Point Symbolic Representation

label written on the task boxes of the task stage). Any exceptions to the above-mentioned general rules are indicated, wherever deemed to be necessary, in the documentation.

2.3.2.2 Task Signal Collector and Task Driver Drawings

The Control sequence for many arithmetic instructions (e.g. DIV (Division), and CVD (Convert to Decimal) etc.) is composed of many control subsequences (Procedures and Subroutines). Many of these control subsequences form separate logic drawings. Task signals with the same functional name (and hence significance) may appear on different drawings of the sequence because they are made active quite often at different time instants in the control sequence. All these task signals with the same functional significance and appearing more than once in the control sequence for any arithmetic instruction are logically ORed together on the Task Signal Collector (TSC) drawings. Since all the task signals with the same functional significance from different sequences will have to be logically combined into one single control line to provide the final control signal to the processing hardware to do the task, it is better to group these signals first locally within each sequence and then from these various groups into the final control signal. The former grouping of signals within the same sequence is done on Task Signal Collector Drawings and later grouping to generate the final control signal is done on Task Driver (TD) Drawings. For example the drawings 370-12, -13, -14 and 384-06 are the TSC drawings for control sequences DIV and CVD and 388-01, -02, -03 are the TD drawings. Thus, the inputs to the TSC drawings are output signals from control point drawings and their outputs go to the input of TD drawings.

The outputs of TD drawings are cable driven to the input of the processing hardware to perform the required task. This hierarchical grouping of signals provides a modular way of collecting signals, and in case of future necessity, other signals can be easily added.

The hardest part in designing such drawings is partitioning the output control signals on the control point drawings among the TSC drawings so that the total number of pins used on these drawings or cards is minimal. To do so it is necessary to place the grouping logic for signals, which are often turned on by the same control point task signal control lines, together on the same card or TSC drawing. This allows one input pin on a card to be used for turning on more than one functional control signal. Unfortunately this can be an extremely time consuming process.

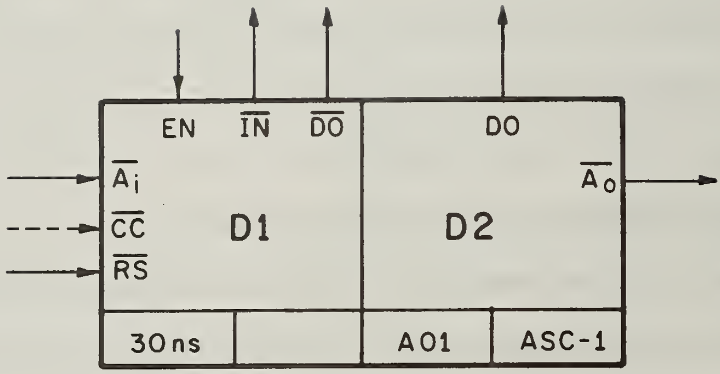
2.3.3 Control Logic Drawing Conventions and Notations

This section describes the symbolic notation used for the representation of a control point, the signal name conventions for the output of both the control point conditional task logic and the sequence stage logic. In addition, the signal name conventions for the "Task Signal Collector" and "Task Driver" drawings are also described.

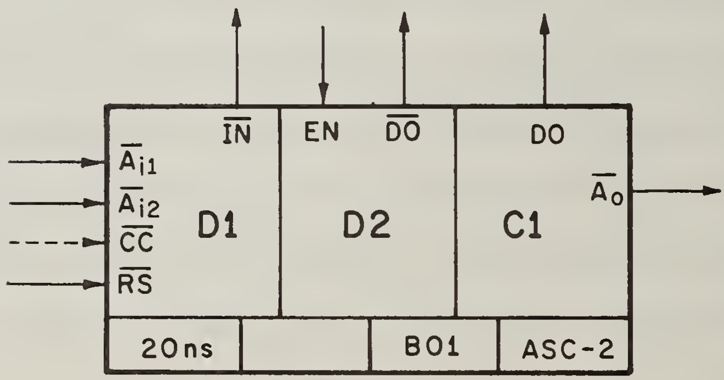
2.3.3.1 Control Point Standard Symbolic Representation

In this paper, a control point is understood to consist of not only the memory element and the timing stage (as in Martin's Case), but also the logic associated with the feedback path between the output of timing model and the reset input of the memory element (Figure 2.3.2.1.1). For ease of use, the combination of memory element and the basic timing model is represented on the main logic drawings by one of the symbol used in Figure 2.3.3.1.1. Note that the various input and output signals are indicated by letters at the points where they enter and leave the box. The upper part of the symbol is divided into either two or three boxes (depending on the control point variant being used) which represent the chips used to implement the control point logic. The letters at the center of each box represent the location of these chips on the printed circuit board.

The four boxes in the lower part of the symbol are used to describe parameters for the control point. The left most box contains the control point delay in nanoseconds. The second box is unused at present. The third and fourth boxes contain the control point variant name and the unique name for that point respectively. The variant types are described by a 3 or 4 character



(a) Single Input Control Point



(b) Two Input Control Point

Figure 2.3.3.1.1. Control Point (Without Memory Element Reset) Standard Symbolic Representation

code in which the first letter represents the general type of control point, the second and third numbers represent the variant number and the last letter (always a D, if it is present at all) indicates that a speed up diode is used.

The unique name associated with the control point is the same as that associated with the task stage (or its various task boxes) it is used to implement. Accordingly, the name has the format XYZ-i were XYZ is the name abbreviated from the sequence or subsequence name (to give some functional significance to the control point) and i signifies that it is the ith control point (hence the task stage) out of a totality of control points used to implement that sequence or subsequence. In a few cases, four letters are also used to indicate the name.

$\overline{\text{IN}}$ signifies Indicator. This output from the control point is used for monitoring the Active/Inactive status of the control point. It is very useful for check-out and/or for display at the engineering console.

On the logic drawings, the inputs $\overline{\text{CC}}$ (Common Clear) and $\overline{\text{EN}}$ (Enable) to the control point are now shown on the standard symbol for the control point although they are supposed to be present. However, EN is shown specifically in those cases where an external conditional logic is attached to this input of the control point.

2.3.3.2. Control Point Signal Name Conventions

Associated with each control point (including conditional task logic) are output control lines which carry control signals, input lines carrying task conditions and/or variables making up the task condition (external conditions),

the Advance-in and Advance-out signals and the indicator signals. All the control lines except those carrying external conditions have the name of the corresponding control point associated with it.

A task control line carrying one or more control signals is labelled XYZiTn where XYZi is the name of the control point, T stands for task and n indicates that it is the nth task control line associated with that control point. In case of DETAG control point, the delayed task control line is labelled as XYZiDTn where XYZi and n have the same meanings as before and DT stands for delayed task signal. Note that one control line may activate quite a few control signals and the functional names (abbreviated from the type of control function they perform) of these control signals (which are the same as written in the task boxes of the task stage) are also written on the control line. This helps in identifying the functional (in the control sense) significance of the particular control lines. In general, there are as many control lines as there are task boxes in one task stage. However, if on the same logic drawing two or more control points have control lines carrying the same functional control (task) signals, the control lines are logically-ORed together to form only one output control line so as to make better use of available pins on the control logic card.

The input control line directly connected to the Advance-in input of the control point carries the functional name XYZiAI where XYZi is the name of the control point. Nothing specific can be said about the control line name because in general it can come from different parts of the control logic. Quite often it will come from the sequence stage of other control points.

Each control point has an indcator output $\overline{\text{IN}}$. This indicator control line (via a driver) is labelled XYZNi where XYZi is the control point name and N stands for the indicator. Note that the numerical part (i) of the name appears at the end of label unlike the task control line label.

2.3.3.3 Sequence Stage Signal Name Convention

Following each control point, there is the sequence stage which is essentially combination logic. The input to this combinational logic are the variables making up the sequence condition and the Advance-out signal of the control point. If the output of the sequence stage is connected to some logic on the same logic drawing, then this output is not labelled. However, if it goes to an output pin on the card (because it is connected to a control point on some other drawing and hence PCB) then the corresponding control line is identified as XYZisn where XYZi is the control point name, s stands for sequence stage and n indicates that it is the nth sequence control line for that sequence stage. The functional name associated with the sequence control line is either the name of the Procedure which it initiates or a name with a format XYZiAI where XYZi is the name of the control point which it primes and initiates.

2.3.3.4 "Task Signal Collector" and "Task Driver" Drawings' Signal Name Convention

The input signals to the TSC drawings are the control point task signal control lines and hence carry the label format XYZiTn whose interpretation is explained earlier. The output signal on the TSC drawings are identified by the functional name of the signal followed by a slash and either of the

letters A, B, C,D or E. Each of these five letters identifies the TSC drawing signals as being the result of logical OR of control point task signal control lines from a different set of drawings. For example the letter B identifies the signal as belonging to the set of drawings 370-On (n = 1, 2, ... 12) for DIV instruction control sequence. These letters are necessary to uniquely identify electrically distinct nodes although logically they may be the same. However, in those cases where one physical node on TSC drawing activates, simultaneously, functionally distinct signals, this node carries a label with a format ZYZTDn, where XYZ stands for the sequence name and n is a numeric (n = 1, 2, ...), whose value signifies that it is nth signal node in the TSC drawings for that sequence, for example, the signal node $\overline{CVDTD_1}$ on TSC Drawing No. 384-06. Just as in control point drawings, the functional names of all the signals associated with this signal node are written on the line.

The inputs to the Task Driver drawings are the output signals from TSC drawings and hence carry the label XYZTDn. The output signal nodes of the Task Driver drawings are labelled the same as the corresponding input nodes of the processing hardware to which they are to be connected. These labels are the functional names of the control signals. For an example, see the output node labels on Drawings 388-01, -02, -03.

3. CONTROL SEQUENCES, CONTROL-POINT FLOW CHARTS and THEIR OPERATIONAL DESCRIPTION

3.1 Introduction

This section describes the control sequences for the execution of various arithmetic orders. These control sequences have been divided into subsequences for ease of description, comprehension and modularity. Each control subsequence is described in terms of 'Control-Point' flow chart which shows the various control signals, their time order and the conditions under which they would be activated.

Each control sequence is first represented by a 'Global Flow (GF) diagram'. This GF diagram shows the various control subsequences which make up that control sequence and grossly shows the flow of control from one subsequence to another. An overall gross description is given of the control sequence in terms of the functional significance of the various subsequences that make up the global flow diagram.

Next, a detailed operational description of each subsequence is given in terms of the control point flow chart and the actions performed by the various stages of the control point flow chart.

Still another subsection entitled "Logic implementation" shows the logic drawing number on which the actual logic hardware implementation is shown.

To understand the control sequences, one needs to be familiar with the control algorithms for the execution of corresponding instructions. In this report, the operational description of control algorithms is not given in detail. DCS Report No. 418, which describes the simulation of the control algorithms at the hardware level, gives a description of the control algorithms and the motivation for the various steps in the algorithm. The present report

assumes that the reader is familiar with the contents of Report No. 418, and it is recommended that the two reports should be read concurrently. Since the simulation described in Report No. 418 was done at the hardware level, there is an almost one-to-one correspondence between the simulation flow charts and the control flow charts. However, there are certain differences which can be pointed out in general.

a) Simulation was performed on IBM/360 which is basically a sequential machine. However, in hardware control realization, many of the functions could be done in parallel which in simulation flow charts are shown as being serial. This fact is pointed out whereever deemed necessary in the documentation.

b) All gate functions and shift logic functions are simulated as procedures, whereas in actual hardware, they are realized by activating a 'Select' signal which selects the proper data bus followed by a 'Load' signal. As an example, consider LHR4UH. This means that contents of LH register is right shifted four bits and transmitted to UH register. In simulation flow charts, it suffices to write LHR4UH, whereas in control flow charts, we have to first turn on SLHR4UH which sets the selector flip-flop to select the shifted data to the input of the register UH, and then the gating signal LDUH is turned on which sets the register UH according to the contents of the bus.

c) Illiac III AU contains four cascaded signed-digit subtractors (SDS). In AU simulation, one SDS is simulated as a subprogram and is called four times in succession to simulate the SDS array. In the control logic, the whole Subtractor/Adder array is activated at the same time in parallel.

Any other marked differences between the simulation flow charts and the control flow charts are indicated in place whenever considered necessary to avoid confusion.

Besides describing the various control sequences in terms of their flow charts, the two other subsections describe briefly some other hardware which complements the hardware which implements the control point flow charts. These are the final level task drivers discussed in Section 3.10.3 and other miscellaneous hardware like electronic pushbutton selector switches, power turn on and 'clear and reset' logic, etc.

Before moving further to study the control point flow chart, it should be borne in mind that the AU control logic, as implemented here, assumes that the floating point operands sent by the processor to the AU are normalized to base 16. It is TP's responsibility to guarantee this before transmitting operands to the AU. If the processor fails to do so, the AU control logic might hang up.

Also, it is assumed that TP will not send any arithmetic order which involves the processing of decimal operands except the instructions CVL (convert to long-fixed point) and CVF (convert to floating). It is necessary because none of the arithmetic orders ADD, SUB, CPRA, MPY, DIV, with decimal operands is implemented as of present.

Finally, it should be noted that the arithmetic order POLY for polynomial evaluation is not implemented either.

3.2 INSTRUCTION DECODING

3.2.1 Instruction Variant and Number Type Decoding

The order to be executed by an arithmetic unit is indicated by the four bits of the Instruction Variant (IV) together with the two bits of Number Type (NT) code. Bits number 4-7 of the V-BUS are gated into the IV-Register under control of the signal IVDIVR. The true outputs of the IV-Register flip-flops are designated IV_1 - IV_4, corresponding to V_4 - V_7, respectively.

Similarly, bits number 8 and 9 of the V-BUS are gated into the NT-Register under control of the signal NTDNTR. The true outputs of the NT-Register flip-flops are designated NT_1 and NT_2, corresponding to V_8 and V_9.

The true and complement outputs of the NT-Register drive the NT-Decoder to produce both the true and complement of the following signals.

$$SFIX = \overline{NT_1} \cdot \overline{NT_2}$$

$$LFIX = \overline{NT_1} \cdot NT_2$$

$$FLT = NT_1 \cdot \overline{NT_2}$$

$$DEC = NT_1 \cdot NT_2$$

$$FIX = SFIX \vee LFIX = \overline{NT_1}$$

The logic for the IV- and NT-Registers and Decoders is shown in Drawing AUO-07-321-01.

The true and complement outputs of the IV-Register drive the IV-Decoder which produces the true value of the following signals:

$$\mathrm{CVL} = \overline{IV_1} \cdot \overline{IV_2} \cdot \overline{IV_3} \cdot \overline{IV_4}$$

$$\mathrm{CVL} = \overline{IV_1} \cdot \overline{IV_2} \cdot IV_3 \cdot \overline{IV_4}$$

$$\mathrm{CVD} = \overline{IV_1} \cdot \overline{IV_2} \cdot IV_3 \cdot IV_4$$

$$\mathrm{ADD} = IV_1 \cdot \overline{IV_2} \cdot \overline{IV_3} \cdot \overline{IV_4}$$

$$\mathrm{SUB} = IV_1 \cdot \overline{IV_2} \cdot IV_3 \cdot \overline{IV_4}$$

$$\mathrm{CPRA} = IV_1 \cdot \overline{IV_2} \cdot IV_3 \cdot IV_4$$

$$\mathrm{MPY} = IV_1 \cdot IV_2 \cdot \overline{IV_3} \cdot \overline{IV_4}$$

$$\mathrm{POLY} = IV_1 \cdot IV_2 \cdot \overline{IV_3} \cdot IV_4$$

$$\mathrm{DIV} = IV_1 \cdot IV_2 \cdot IV_3 \cdot \overline{IV_4}$$

$$\mathrm{ASC} = \mathrm{ADD} \vee \mathrm{SUB} \vee \mathrm{CPRA}$$

In addition, a signal to indicate the arrival of a coefficient (other than the first) for a POLY order is generated. It is defined as follows:

$$\mathrm{POLYCO} = IV_1 \cdot IV_2 \cdot IV_3 \cdot IV_4 \cdot \mathrm{POLIP}$$

where POLIP = POLY in Progress.

Note that not all bit patterns of IV and NT correspond to legal instructions. If an invalid pattern is sent to the AU, an ILLØP (Illegal Operation) condition is generated. A map of the ILLØP condition is given in Table 3.2.1.

Mnemonic	Binary Code	Decimal Code (IV \ NT)	SFIX 0	LFIX 1	FLT 2	DEC 3
	0000	0	1	1	1	1
CVL	0001	1	1	1	0	0
CVF	0010	2	0	0	1	0
CVD	0011	3	0	0	0	1
	0100	4	1	1	1	1
	0101	5	1	1	1	1
	0110	6	1	1	1	1
	0111	7	1	1	1	1
ADD	1000	8	1	1	0	1
	1001	9	1	1	1	1
SUB	1010	10	1	1	0	1
CPRA	1011	11	1	1	0	1
MPY	1100	12	0	0	0	1
POLY	1101	13	1	1	0	1
DIV	1110	14	0	0	0	1
POLY	1111	15	1	1	1	1

- = don't care

Table 3.2.1 - Map for ILLØP Conditions

The most efficient implementation is realized by writing an expression for the "0's" in the table. Thus, if we represent the IV-string by its decimal equivalent (for example 0001 = (1)), then

$$
\begin{aligned}
\text{ILL}\emptyset\text{P} = \;& (1) \cdot \overline{\text{FIX}} \\
& v(2) \cdot \overline{\text{FLT}} \\
& v(3) \cdot \overline{\text{DEC}} \\
& v(8) \cdot \text{FLT} \\
& v(10) \cdot \text{FLT} \\
& v(11) \cdot \text{FLT} \\
& v(12) \cdot \overline{\text{DEC}} \\
& v(13) \cdot \text{FLT} \\
& v(14) \cdot \overline{\text{DEC}}
\end{aligned}
$$

3.2.2 Decimal Operand Sign Register and Decoder (DOSIRED)

In case of decimal operand instruction, bits numbered 15-18 of the V-BUS are gated into the Decimal Operand Sign (DOS) Register under control of the signal LDDSIGNA. The true and complement outputs of the DOS-Register drive the DOS-Decoder which produces two signals, $\overline{\text{SSIGNAPOS}}$ and $\overline{\text{SSIGNANEG}}$, for positive and negative sign of the operands respectively. These signals in turn set the sign flip-flop, SIGNA (Drawing AUO-390-07-08), of the decimal operand A to the positive and negative state respectively. In case the given sign code does not conform to the USASCII code, possibly due to some error, then an illegal decimal operand signal ILLD∅P becomes true. This is shown in drawing AUO-07-321-01.

3.2.3 Logic Implementation

Logic implementation of Instruction Variant and Number type decoding is shown in control drawing AUO-07-321-01. Also shown on the same drawing is Decimal Operand Sign Register and Decoder (DOSIRED).

3.3 V-BUS INPUT OPERAND LOAD (VIN) CONTROL SEQUENCE

Upon rapid initial decoding of the instruction variant and number type, VIN control sequence loads the operands as received, one word at a time, into appropriate registers. After all operands have been loaded, the control of execution is switched to an appropriate control sequence according to the instruction variant. After the execution of that control sequence is completed, control is switched back to VIN for the next arithmetic order.

3.3.1 Global Flow Description

Figure 3.3.1.1 shows the global flow diagram for VIN control sequence. VIN control sequence is composed of four control subsequences, some or all of which are entered depending on the type of instruction variant and number type. The operand is received one word at a time, in the following order: left word of operand A, left word of operand B, then the right word of operand A, and finally the right word of operand B. Figure 3.3.1.2 shows the order of word transmission from TP to AU in the case of various number types.

Subsequence FIWLOB loads the instruction variant and number type registers from the control byte of V-Bus Input word, decodes the instruction and then loads the first word of the operand into appropriate registers in the AU processing hardware. If there is any parity error or the instruction decoding logic indicates an illegal operand, then an appropriate exit is taken. Since the instructions CVD and CVF involve unary operation and single word operand, the control sequence branches to CVD and CVF subsequences and in all other types of instructions, the control sequence goes to SEWLO and BRAIN to load the next word of the operands.

SEWLO loads either the second word of the double word operand for unary operations (e.g., number type conversion instructions for decimal and floating point operands) or the first word of second operand in case of other instructions. BRAIN causes the control sequence to branch to an appropriate control sequence, for example, to CVL, CVD and CVF in case of floating and decimal operands and unary instructions, and to control subsequences MPY and DIV for fixed point operants. For binary instructions require double word operands, VIN control goes to TIFOWLOB.

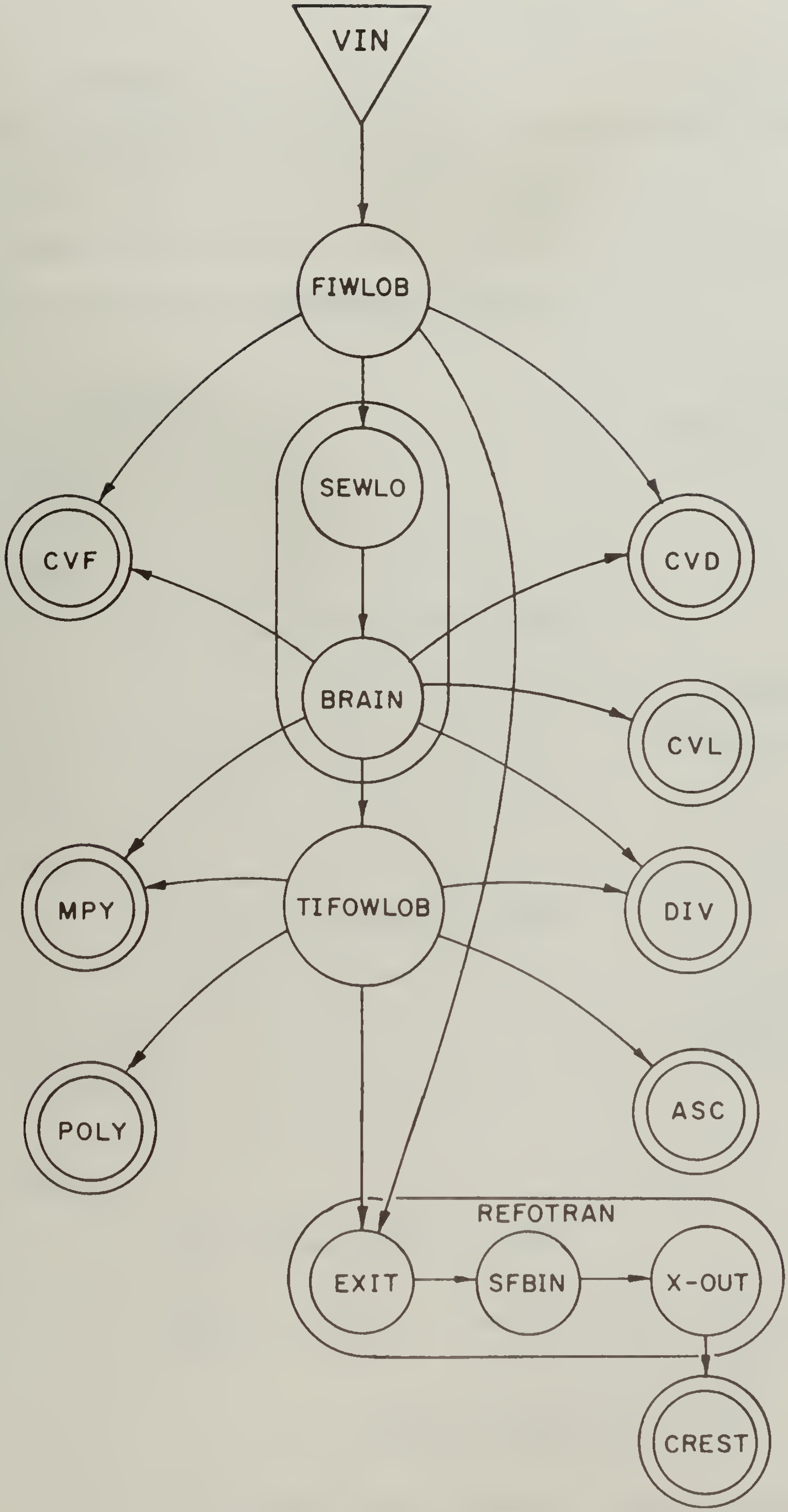


Figure 3.3.1.1. Global Flow Diagram for "VIN--Input Operand Load from V-Bus and Branch to Appropriate Arithmetic Order" Control Sequence

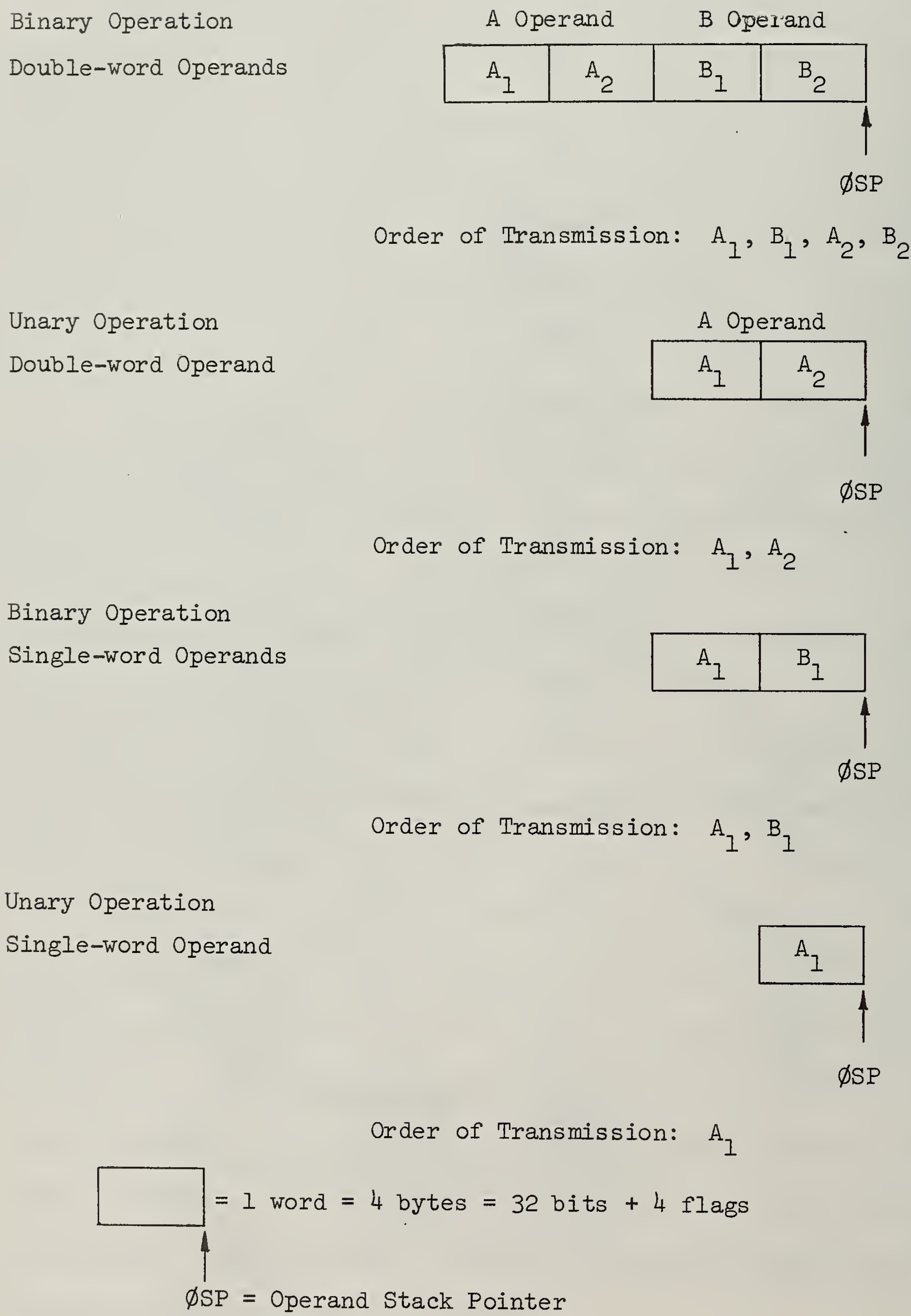


Figure 3.3.1.2. Order of Word Transmission to AU

TIFOWLOB loads the second words of operands A and B into appropriate registers, checks for the parity errors and then branches to an appropriate control sequence depending on the instruction type as decoded in FIWLOB.

It may be noted that the control sequence for arithmetic instruction POLY has not been implemented and for the present it has been decided that the TP would not send any POLY order to the arithmetic unit.

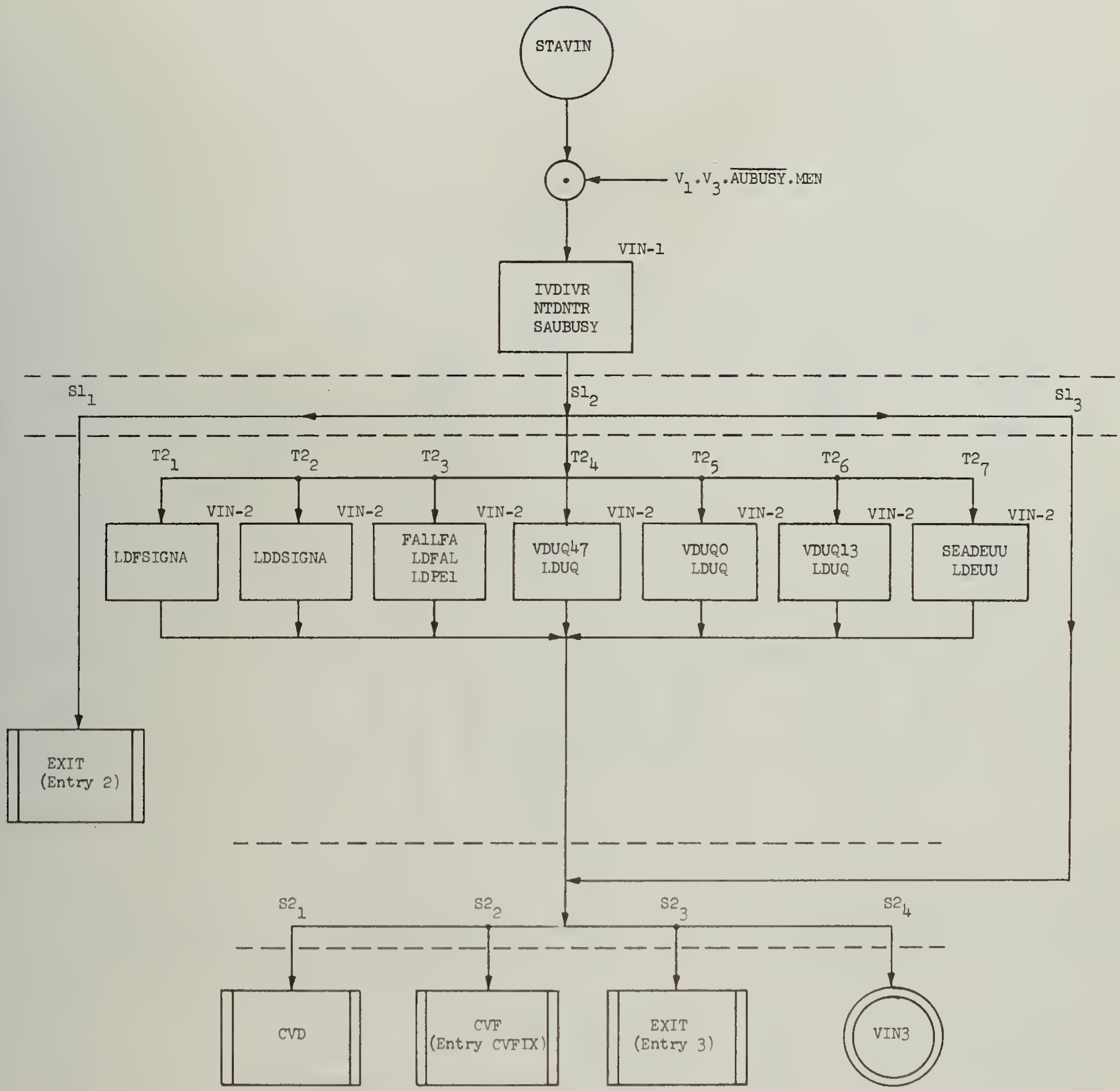
3.3.2 Control-Point Flow Description

Figures 3.3.2.1, 3.3.2.2 and 3.3.2.3 show the control point flow charts of various subsequences used in VIN control sequence.

Task stage VIN-1 loads the IV-Register and NT-Register and sets the AUBUSY indicator.

Branch $S1_1$ in sequence stage S1 should be carefully noted. This branch is taken whenever an illegal instruction variant or number type is indicated by the instruction decoder. The words from the V-Bus are transferred to appropriate registers depending upon the instruction and the control sequence will operate only if the various task and sequence conditions are satisfied. Whenever the instruction variant and/or number type received by the control are invalid, the control sequence flow will hang up if a branch is not taken to the EXIT control subsequence. In the EXIT subsequence, the control flow must wait for a duration necessary for the double-word binary operands transfer which is the maximum time TP ever takes in transferring data to the AU,before the AU calls back the TP to inform it of an invalid command.

In the simulation flow charts, a check for parity error is not specifically indicated whereas the control-point flow charts do show a parity check in the form of loading the parity error flip-flop (LDPEi, i = 1-3). The rest of the control-point flow chart is almost identical to the simulation flow charts and should not cause any confusion.



$S1_1$ = ILLØP

$S1_2$ = $\overline{(\text{ILLØP v POLYCO})}$

$S1_3$ = POLYCO

$T2_1$ = $\overline{\text{DEC}}$

$T2_2$ = DEC

$T2_3$ = NONE

$T2_4$ = FIX.(CVD v MPY)

$T2_5$ = FIX.(CVF v DIV)

$T2_6$ = $\overline{\text{FIX}}$ v (CVF v DIV)

$T2_7$ = FLT

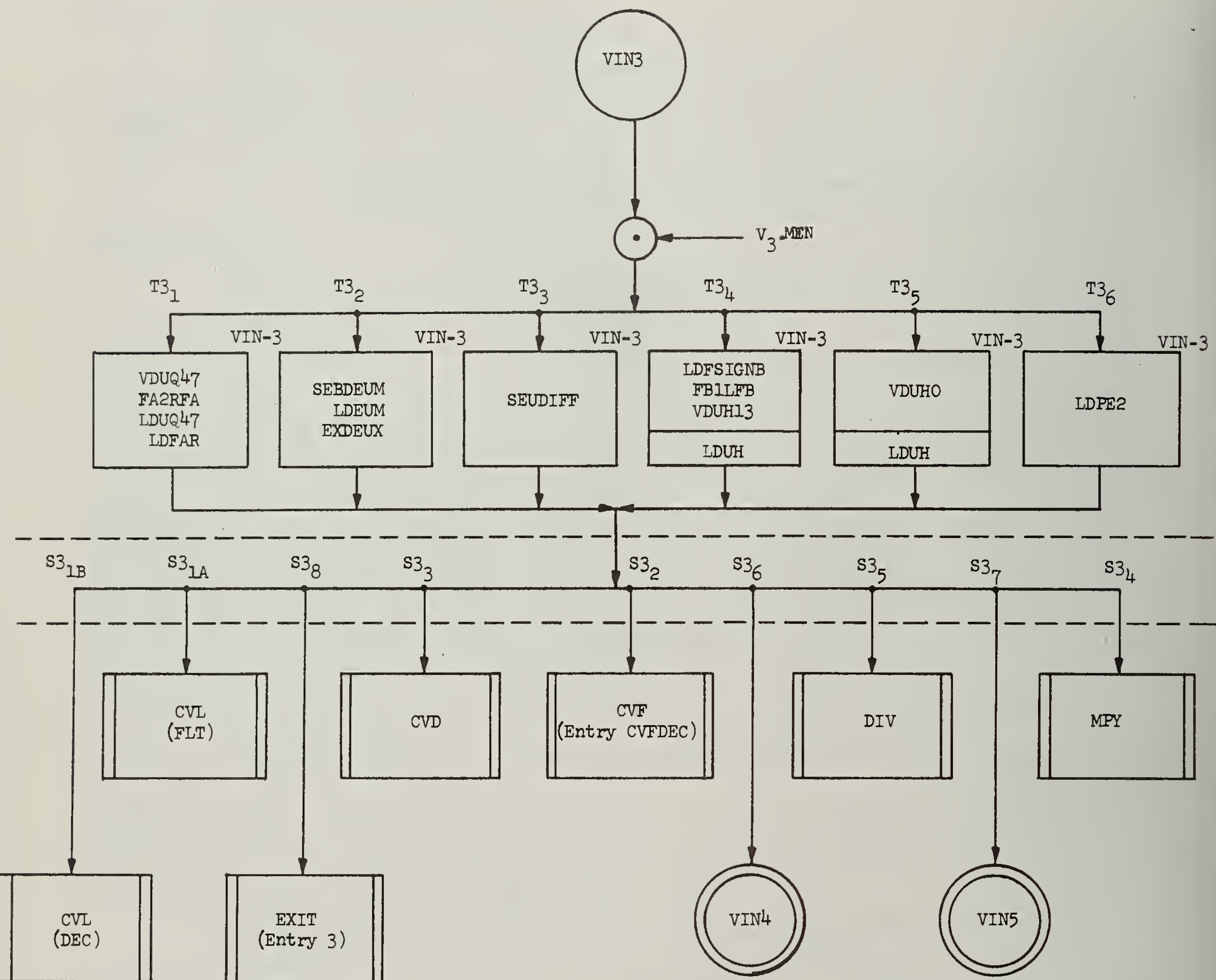
$S2_1$ = CVD.FIX.$\overline{\text{PERR}}$

$S2_2$ = CVF.FIX.$\overline{\text{PERR}}$

$S2_3$ = PERR.(CVD.FIX v CVF.FIX)

$S2_4$ = $\overline{(\text{CVD.FIX v CVF.FIX})}$

Figure 3.3.2.1. FIWLOB—First Word Load and Branch



$T3_1 = (\overline{IV_1} \text{ v DIV.FIX})$

$T3_2 = IV_1.\overline{FIX}$

$T3_3 = FLT.\overline{MPY}$

$T3_4 = IV_1$

$T3_5 = FIX$

$T3_6 = \text{(NONE)}$

$S3_{1B} = CVL.DEC.\overline{ILLD\emptyset P}.\overline{PERR}$

$S3_{1A} = CVL.FLT.\overline{PERR}$

$S3_3 = CVD.FLT.\overline{PERR}$

$S3_2 = CVF.DEC.\overline{ILLD\emptyset P}.\overline{PERR}$

$S3_5 = DIV.FIX.\overline{PERR}$

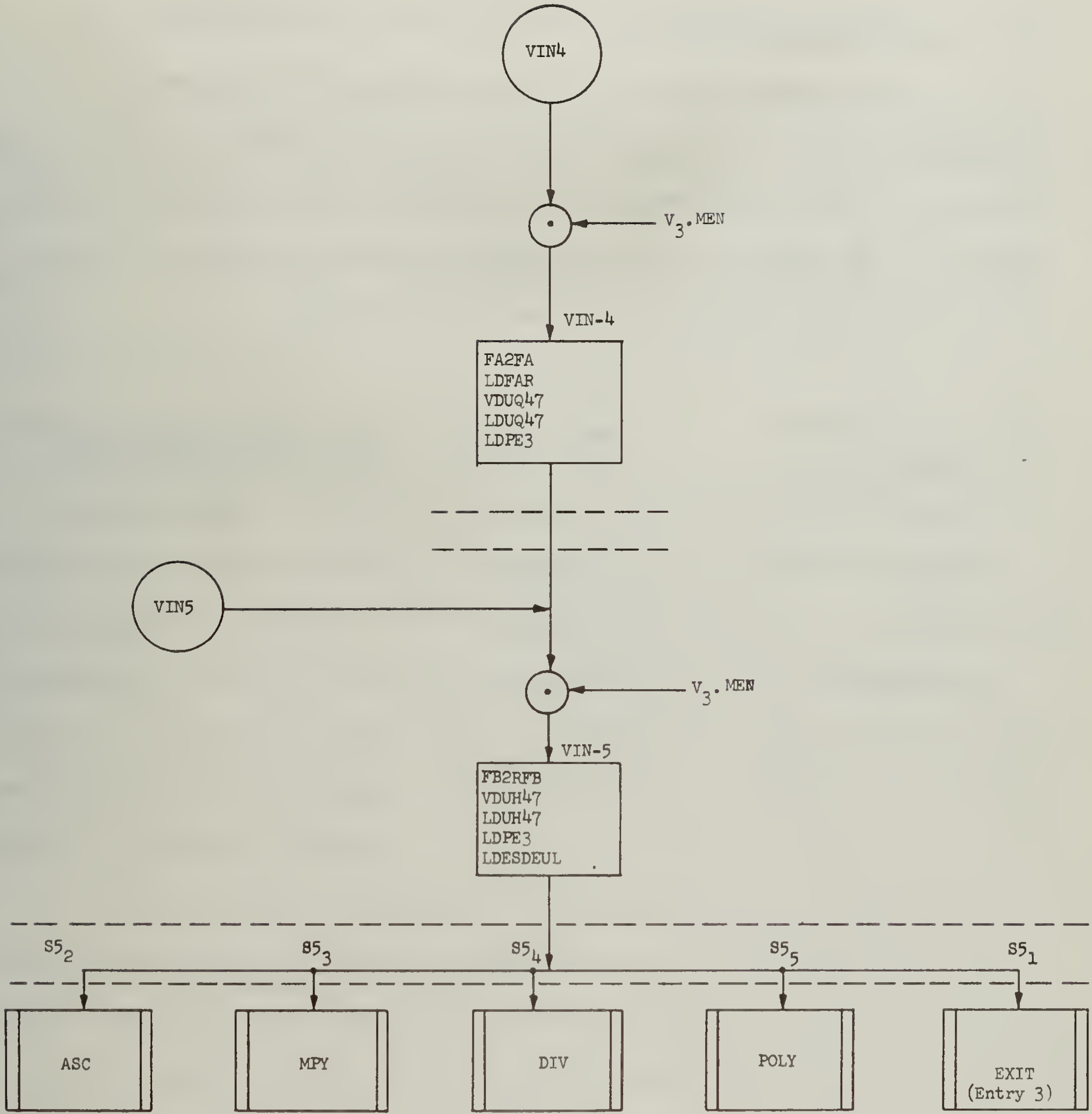
$S3_4 = MPY.FIX.\overline{PERR}$

$S3_8 = (\text{CVL.FLT v CVL.DEC v CVF.DEC v CVD.FLT v MPY.FIX v DIV.FIX) PERR v (CVL.DEC v CVF.DEC) } \overline{PERR}.ILLD\emptyset P$

$S3_6 = IV_1.FLT.\overline{POLYCO}.\overline{PERR}$

$S3_7 = POLYCO$

Figure 3.3.2.2. SEWLO and BRAIN—Second Word Load and Branch to Appropriate Instruction



$S5_1$ = PERR

$S5_2$ = ASC.$\overline{\text{PERR}}$

$S5_3$ = MPY.$\overline{\text{PERR}}$

$S5_4$ = DIV.$\overline{\text{PERR}}$

$S5_5$ = POLY.$\overline{\text{PERR}}$

Figure 3.3.2.3. TIFOWLOB—Third and Fourth Words Load and Branch

3.3.3 Logic Implementation

Logic implementation of control sequence VIN is shown in control drawings AUO-07-332-01 to AUO-07-332-04. The table below shows the names of the various control subsequences, their control point flow chart figure number, and the number of the corresponding drawing on which the control point flow chart is logically implemented in hardware.

Control subsequence name	Figure Number	Corresponding logic drawing no.
FIWLOB	3.3.2.1	AUO-07-332-01
SEWLO	3.3.2.2	-332-02
BRAIN	3.3.2.2	-332-03
TIFLOWLOB	3.3.2.3	-332-04

3.4 RESULT FORMATTING AND TRANSMISSION (REFOTRAN) CONTROL SEQUENCE

This subsequence forms the last part of each control sequence and is used to format the result properly before transmission. The formatting involves the normalization of the result, setting of proper flags and error indicators. The final part of the subsequence transfers the result to the TP via the Exchange Net.

3.4.1 Global Flow Description

Figure 3.4.1.1 shows the global flow diagram for REFOTRAN control sequence. This control sequence is composed of three subsequences which perform different operations. In control subsequence EXIT, either the contents of register UQ, and hence the result, is normalized if no illegal operand (illegal op code and/or parity error) is encountered during processing,or the normalization of UQ is skipped. The normalization of the result is done by calling the subroutine NØRMUQ.

Next the control branches to subsequence SFBIN where overflow/ underflow conditions are tested to appropriately set the exponent of the result to corresponding unique values ($\phi V=1 \Rightarrow EUL$ should be all ones, etc.), and also the flags are set to indicate either errors and exceptional conditions and/or the result (>,=,<) of executing the arithmetic order CPRA.

Now the result is ready in proper format and the control branches to subsequence X-ØUT which causes the transmission of one or two words in the result. After the result transfer is complete, the control goes to sequence CREST which reinitializes the AU control and processing hardware to await the next arithmetic order.

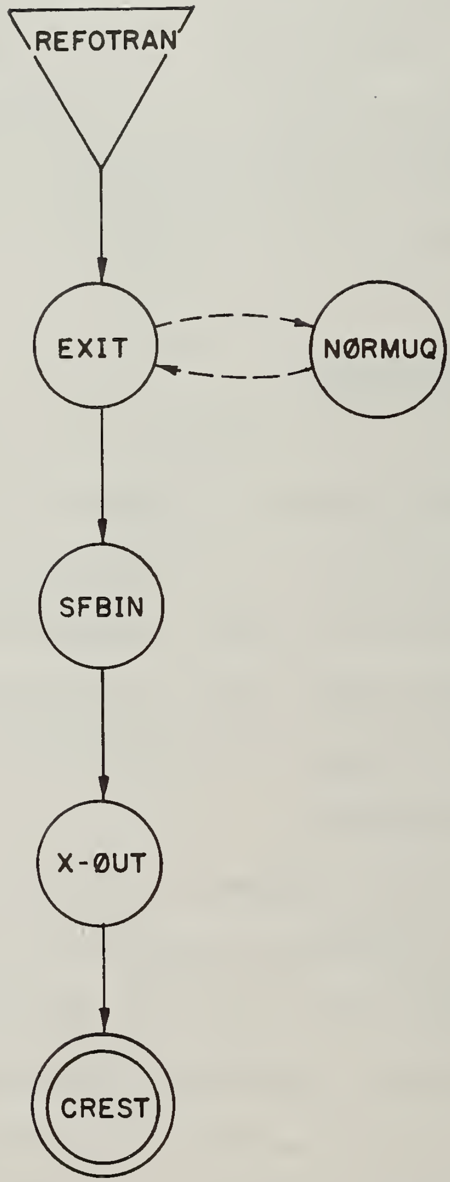


Figure 3.4.1.1. Global Flow Diagram for "REFOTRAN--Result Formatting and Transmission" Control Sequence

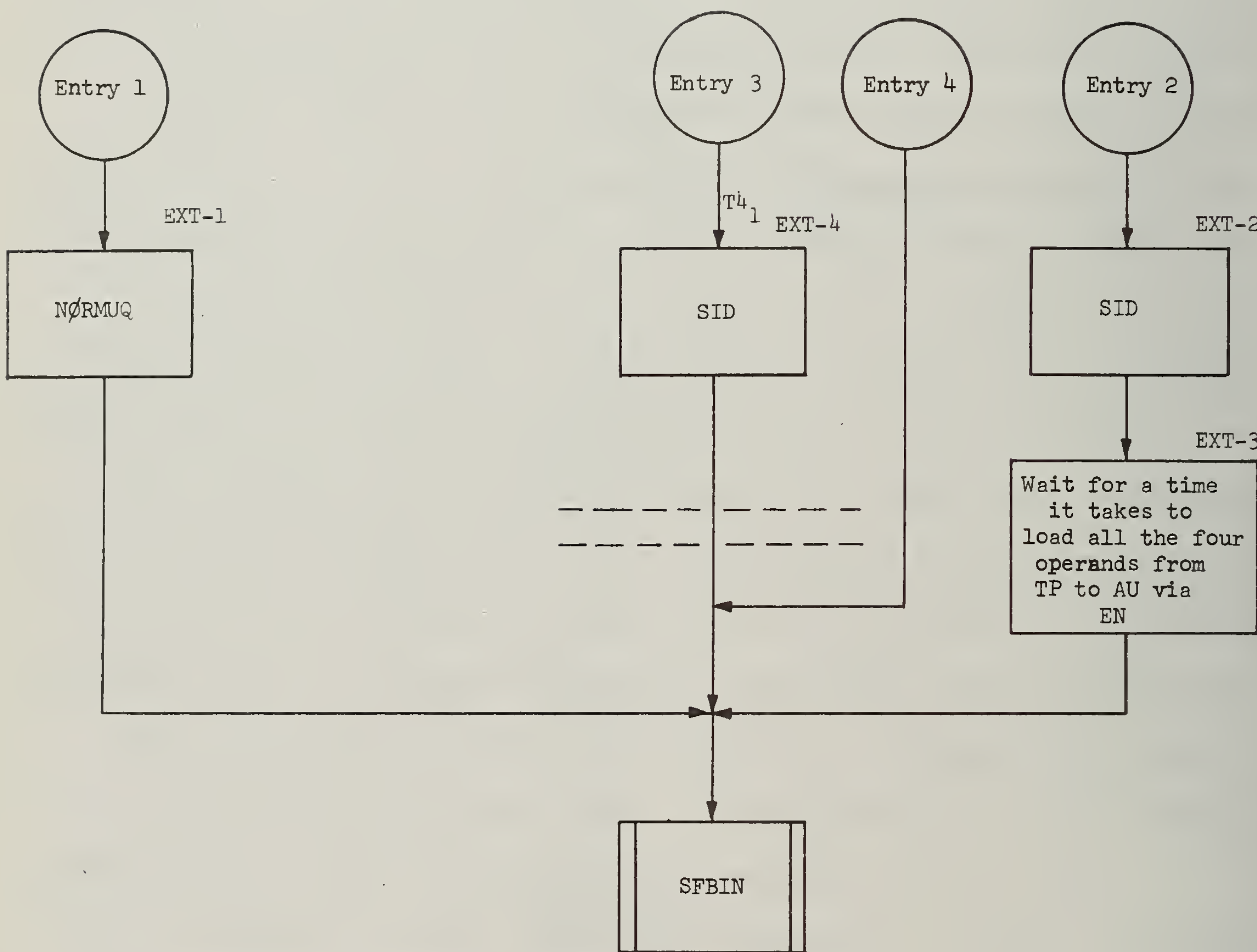
3.4.2 Control Point Flow Description

3.4.2.1 EXIT

Figure 3.4.2.1 shows the control point flow chart for the EXIT control subsequence. It is composed of four task stages and has four entry points. Entry 1 indicates that no illegal data or operand were encountered during the arithmetic order processing, and hence the task stage EXIT-1 calls the control sequence NØRMUQ which normalizes the result which was placed in the register UQ earlier by another control sequence. Entry 2 takes place when an illegal instruction variant or number type is recognized by the VIN control sequence. In such a case, the control would have hung up if an attempt was made to load the operands instead of bypassing the VIN sequence. Task stage EXIT-2 sets a flip-flop ID indicating Illegal Data and control then goes to task stage EXIT-3. This stage is a dummy stage, produces no task control signals except waiting for the worst time it takes for the VIN-Control sequence in any situation. This duration will be found by trial and error. Entry-3 leads to task stage EXIT-4 in which the flip-flop ID is set because an illegal decimal operand (wrong decimal sign code) was pointed out by the decoder. Entry 4 does nothing but is placed here only for uniformity.

After all the task stages have finished their task, the EXIT control subsequence branches to subsequence SFBIN where flags and indicators are set.

Note that there is no corresponding procedure in AU simulation.



$T4_1$ = ILLDØP

Figure 3.4.2.1. EXIT

3.4.2.1.1 <u>NØRMUQ</u>

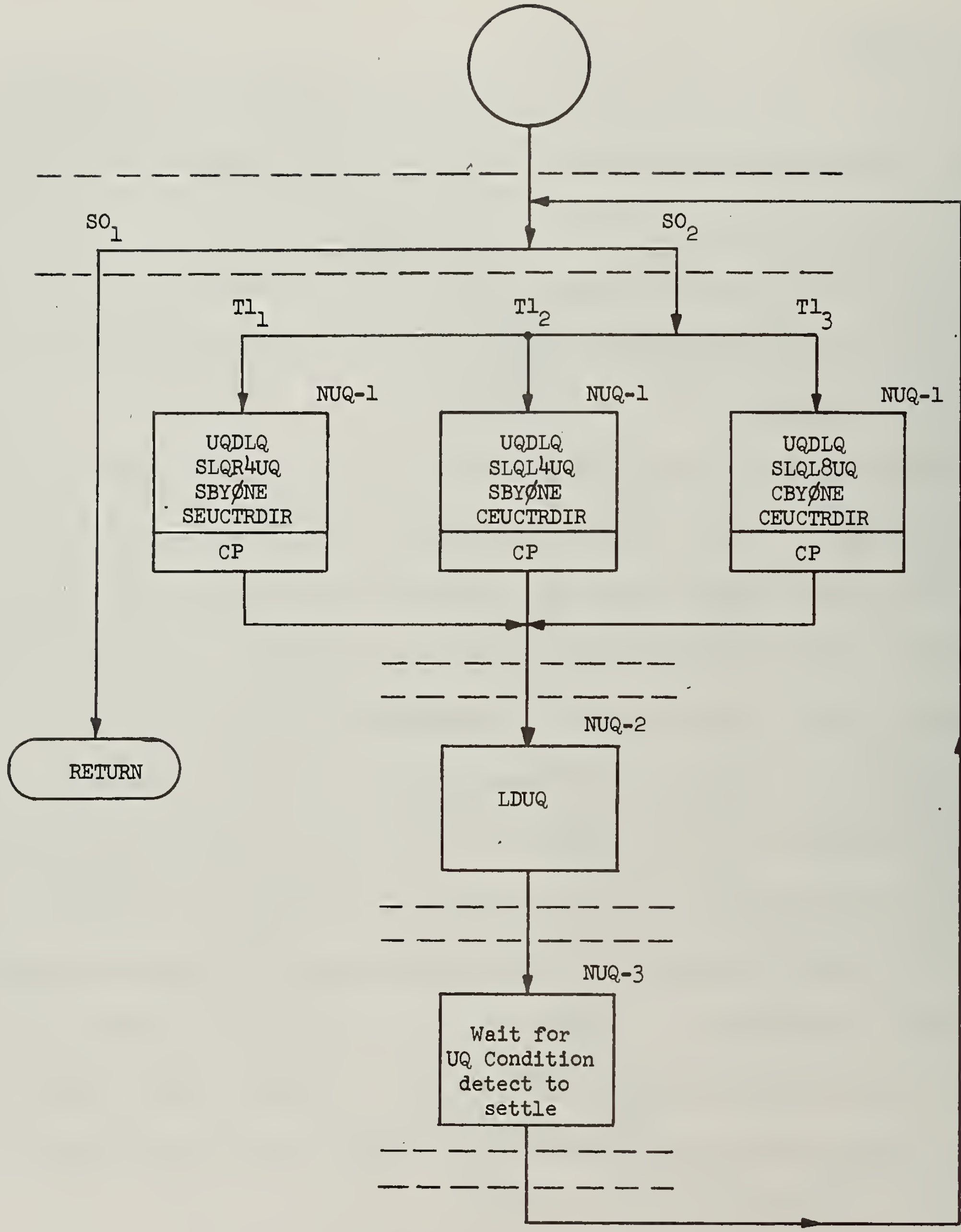
Figure 3.4.2.1.1 shows the control point flow chart for NØRMUQ. This control sequence serves two purposes; one is to normalize the contents of UQ (base 16) such that no more than three leading zeros are retained in the mantissa of the result, and the other purpose is to confine the mantissa to 56 bits so that the leading eight bits of the first word may contain the exponent of the result.

Sequence stage S0 checks if the result, i.e. the contents of UQ, need reformatting. If they do (condition SO_2), the task stage NUQ-1 is entered. Under conditions Tl_2 and Tl_3, the paths are established for left shifting the contents of UQ by 4 or 8 bits, respectively, and at the same time, the exponent in EUL counter-register is decremented by one or two.

Under task condition Tl_1, where the mantissa of the result has overflowed into space for the exponent, the task NUQ-1 sets up the path for right shifting the contents of UQ by 4 bits and increases the exponent by one.

Task stage NUQ-2 gates the shifted contents of UQ into register UQ.

Task stage NUQ-3 is a dummy stage to provide for enough time so that the condition detect logic at the output of register UQ is stable before the control loops back again to see if the control needs to go through another iteration for normalization.



$SO_1 = (\mathrm{UQEZ} \vee \mathrm{UQ58EZ}.\overline{\mathrm{UQ912EZ}})$

$SO_2 = \overline{SO_1}$

$Tl_1 = \overline{\mathrm{UQ58EZ}}$

$Tl_2 = \mathrm{UQ58EZ}.\mathrm{UQ912EZ}.\overline{\mathrm{UQ1316EZ}}$

$Tl_3 = \mathrm{UQ58EZ}.\mathrm{UQ912EZ}.\mathrm{UQ1316EZ}.\overline{\mathrm{UQEZ}}$

Figure 3.4.2.1.1 NØRMUQ—Normalize UQ

3.4.2.2 SFBIN

Figure 3.4.2.2.1 shows the control point flow chart for the control subsequence SFBIN whose function is to Set Flags and Bogus and Arithmetic Indicators. Flag bit designation for arithmetic indicators is given in Figure 3.4.2.2.2 which is reproduced here from Volume 1, DCS Report No. 366. It is recommended that the reader should recapitulate the Sections 1.5 and 1.6 of the Volume 1 (DCS Report No. 366) to fully appreciate the functions of this control subsequence.

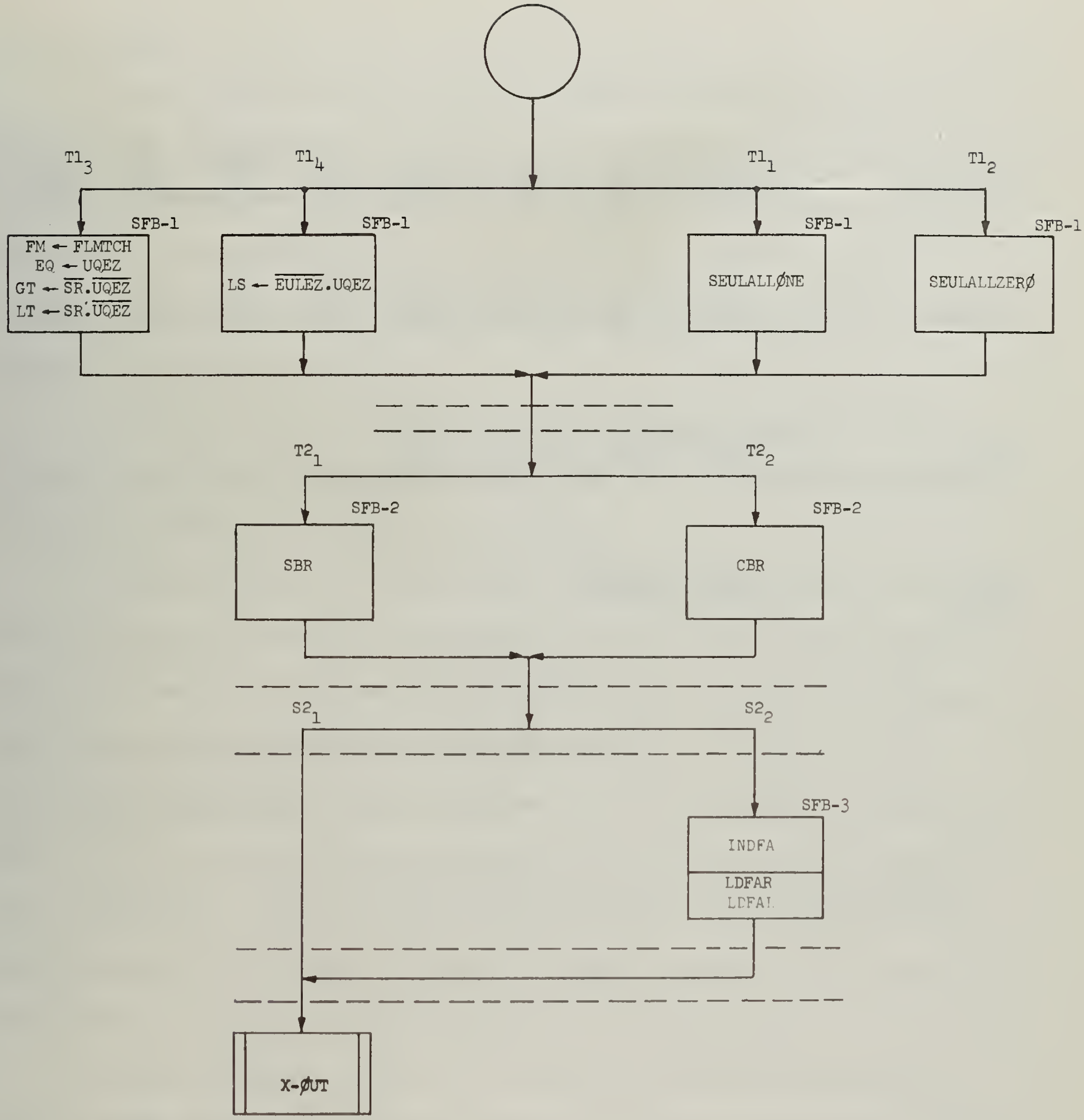
Task stage SFB-1 under task conditions Tl_1 and Tl_2 sets the exponent of the result as +63 and -64--the largest and the smallest values of the exponent. These task conditions indicate the overflow and underflow of the exponent. Task condition Tl_3 applies only for the Algebraic Comparison arithmetic order and sets the arithmetic indicators (flags) according to the result of the comparison. Task condition Tl_4 sets the loss of significance indicator LS when the mantissa of the result is all zero and the exponent is other than zero.

If any of the exceptional arithmetic conditions has occurred during the arithmetic order processing, then task stage SFB-2 sets the Bogus Result (BR) indicator flip-flop. It should be noted that although the flow chart indicates that if the exceptional conditions have not occurred, the flip-flop BR should be cleared (set to zero state), but in actual control hardware, no action is taken because the flip-flop BR is already in zero state, being made so during subsequence CREST or during initialization and power turn on.

The control next enters sequence stage S2 and decides whether the indicator flags should be set or not. When the arithmetic order is Compare Algebraic (CPRA) or if any exceptional arithmetic condition has occurred (condition $S2_2$), the signal INDFA is turned on followed by LDFAL, LDFAR,

which respectively select and load the various indicators into the flag register FA. Finally, the control goes to subsequence X-ØUT which transmits the result to the TP via the exchange net. If the sequence condition $S2_2$ is false, the control directly bypasses to subsequence X-ØUT.

A discrepancy should be noted in the flag bit designations for arithmetic indicators as given in the simulation manual (DCS Report No. 418, p. 41) and as specified in DCS Report No. 366, Section 1.6.1--2/3. But, the latter one is correct because it agrees with the interpretation by the TP. Figure 3.4.2.2 shows the correct flag bit designation for arithmetic indicators.



$T1_5$ = ($\overline{\text{EUAØV v EUAUN}}$)

$T1_3$ = CPRA

$T1_4$ = (ADD v SUB) v POLY.FLT

$T1_1$ = EUAØV

$T1_2$ = EUAUN

$T2_1$ = (ØV v UN v LS v ID)

$T2_2$ = $\overline{T2_1}$

$S2_1$ = $\overline{S2_1}$

$S2_2$ = (BR v CPRA)

Figure 3.4.2.2.1. SFBIN—Set Flags, Bogus and Arithmetic Indicators

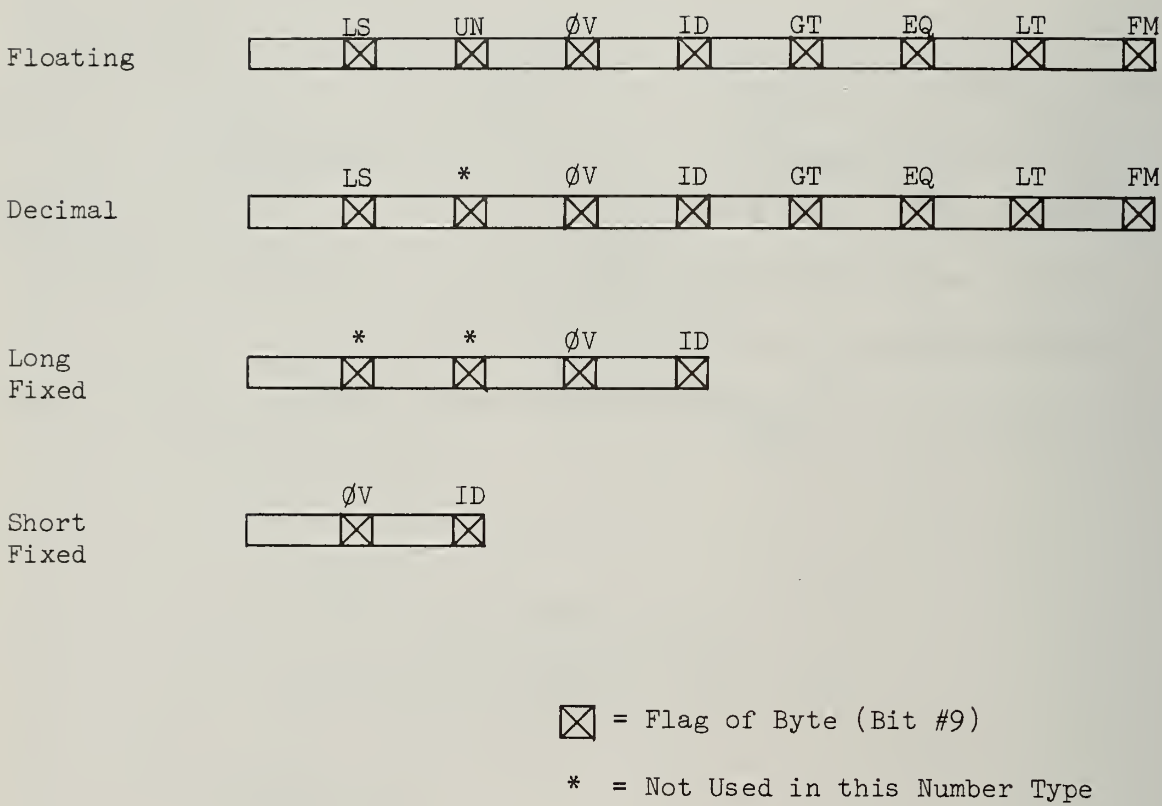


Figure 3.4.2.2.2. Flag Bit Designation for Arithmetic Indicators

3.4.2.3 X-ØUT

Figure 3.4.2.3 shows the control point flow chart for the sub-sequence X-ØUT which performs the function of transmitting the result of arithmetic processing to the taxicrinic processor (TP) which is waiting for the result. This transmission takes place via the exchange net and this subsequence mainly requests and obtains access to the exchange net and then transmits the result to the TP.

Sequence stage S0 examines the ILLØP (Illegal Øperation Code) flip-flop and enters or bypasses the task stage XØUT-1 depending where the flip-flop is in zero or one state, respectively.

When the operation code is legal, the task stage XØUT-1 sets up the data paths for the transfer of the exponent and first three most significant bytes of the result (to XT-bus) if the number type of the result is floating point (task condition $T1_1$); otherwise, the data paths for bytes 0-4 of register UQ which holds the result are set up. In both cases, the flip-flop SCWORD is set to zero state indicating that the first word of the result is being transferred. If the Operation Code is illegal, task stage XØUT-1 is bypassed because UQ is blank as it was not loaded during the VIN sequence, and hence there is no need to set up any paths for data transfer.

Task stage XØUT-2 sets to '1' state the two control flip-flops XTDES and UREN. The control signal XTDES sets up a path for transfer of the result to the exchange net and UREN control signal requests the exchange net access.

The control now transfers to the task stage XØUT-3 where the control waits for a reply from the exchange net, granting access as indicated by $V_2 \cdot \overline{MS}$. As soon as the access is granted, the control task stage XØUT-3 generates a TIØ pulse which is transferred to the processor via the exchange net control

byte and is used by the processor to accept the data from the exchange net. For more details see DCS File No. 790 which discusses "Processor-Unit Communication via the Exchange Net" in more detail.

Sequence stage S3 decides whether both the words of the result have been transferred. If not, as indicated by $S3_1$, the data path for the transfer of second word is set up by setting the flip-flop UQ47DXT to '1' state, and at the same time, the flip-flop SCWORD is set to '1' state indicating that second word is being transferred. The control again loops back to task stage XØUT-3 where the data transfer takes place as described earlier. If the second word has already been transferred, the control exits to task stage XØUT-5 where the flip-flop SCWORD is again set to zero state to make the subsequence ready for further use in the next arithmetic order processing control sequence.

It should be carefully noted here that although the fixed-point type result has only one word of valid data, the subsequence X-ØUT always transfers two words as if the result was always floating point type. It causes no problem as far as the TP is concerned because it can ignore the second word, whereas this way the control is simplified and does not have to use special condition detect logic for fixed point type result.

The control finally branches to control subsequence CREST (clear and reset) which resets every control flip-flop to the initial state, and the control gets ready to accept another arithmetic order from a processor.

<u>NOTE</u>: It should be noted that in case of parity error during VIN sequence, the result transferred out during subsequence X-ØUT is not all zero as mentioned in Section 2.4.2--4/4 of Illiac III Arithmetic Unit, Vol. 1, DCS Report No. 366, but rather the contents of UQ which is one of the operands. It may or may not be the one which has the improper parity.

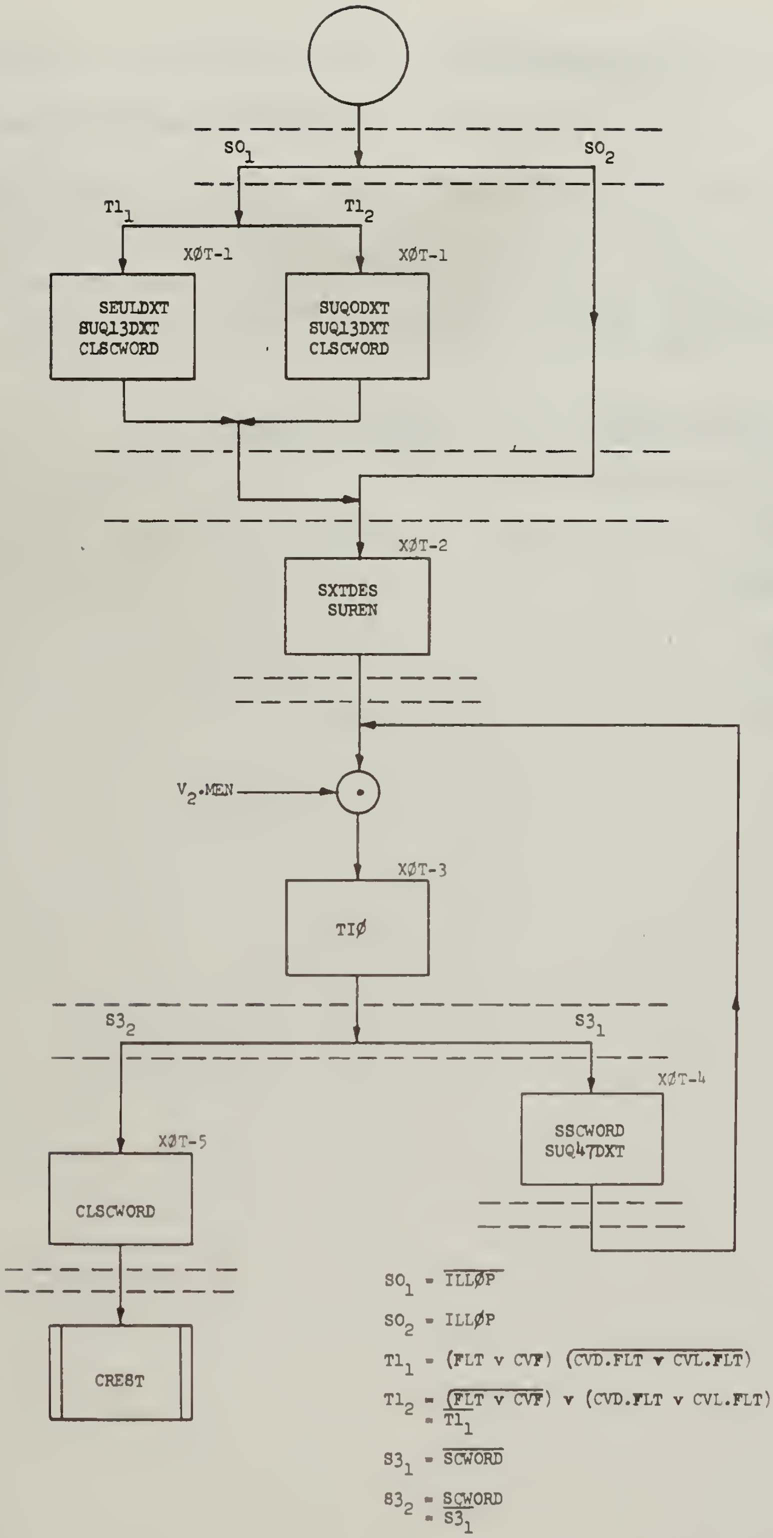


Figure 3.4.2.3. X-ØUT—Transfer Result to Processor via Exchange-Net

3.4.3 Logic Implementation

Logic implementation of REFOTRAN control sequence is shown in control drawings AUO-07-332-05, AUO-07-352-04 through 352-06. The table below summarizes the correspondence between the control point flow charts of subsequences and their logic implementation drawings.

Control subsequence name	Figure Number	Corresponding logic drawing no.
EXIT	3.4.2.1	AUO-07-332-05
NØRMUQ	3.4.2.1.1	-352-04
SFBIN	3.4.2.2.1	-352-05
X-ØUT	3.4.2.3	-352-06

3.5 ADD, SUBTRACT, COMPARE ALGEBRAIC (ASC) INSTRUCTIONS CONTROL SEQUENCE

This control sequence is entered whenever the arithmetic instruction to be executed is any one of Add, Subtract (SUB) or Compare Algebraic (CPRA). All these three instructions have been lumped into one control sequence because their execution use almost the same control subsequences and in essentially the same time order. This control sequence is used for floating point addition, subtraction or comparison. Fixed point addition, subtraction and comparison operations are done in the TP.

3.5.1 Global Flow Description

Figure 3.5.1.1 shows the global flow diagram of ASC control-sequence. Since this control sequence is used for floating point addition, subtraction and comparison, the first order of business is to do the operand alignment. The control subsequence OPACAS examines the algebraic value of the difference of the exponents calculated by the exponent unit processing hardware. It then calls up, using this information, the appropriate operand shifting control subsequence and depending upon whether the sign of the result SR is predictable or not, it sets the SDB (Sign-Digit-Bypass) indicator.

If the exponent unit adder indicates either an overflow/underflow or if the difference of exponents is $> |13|$, which means that one operand is insignificantly small compared to the other and consequently the control sequence goes to subsequence X-ØUT which transfers the result (the bigger of the two operands) to the TP via the exchange net. However, if the difference of the exponent is < 13, the control branches to control subsequence CAL. CAL subsequence provides the control signals to the Adder/Subtractor array which performs the actual addition or subtraction depending on the instruction variant. Subsequence CAL makes use of another subsequence ASIM in order to convert the output of SDS-array into the conventional form.

After the result is calculated, the control branches to subsequence EXIT which calls upon subsequences NØRMUQ and SFBIN to normalize the result and set the flags and other indicators respectively. Now the result is ready to be transmitted to the TP and thus control switches over to control subsequence X-ØUT and after the result is transferred to the TP, the control finally goes to subsequence CREST where the control logic is reset to await for the next instruction.

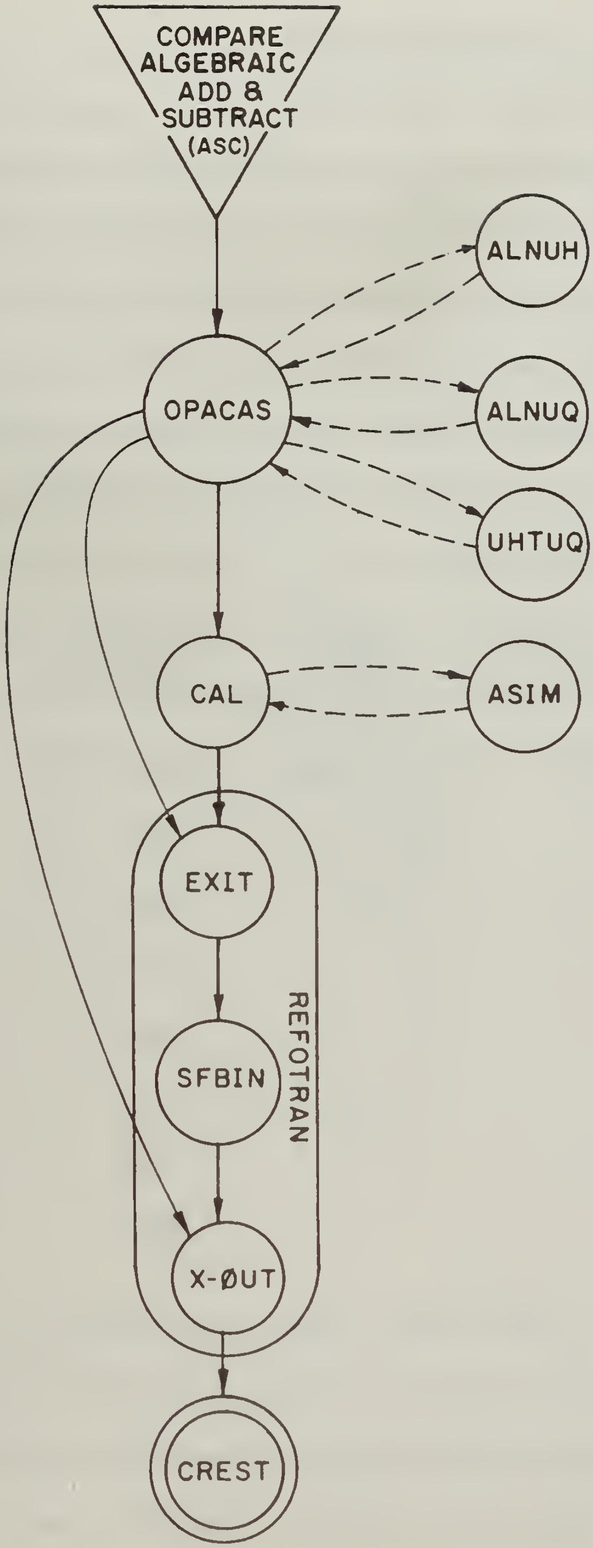


Figure 3.5.1.1. Global Flow Diagram for "ASC--Add, Subtract and Compare Algebraic" Control Sequence

3.5.2 Control-Point Flow Description

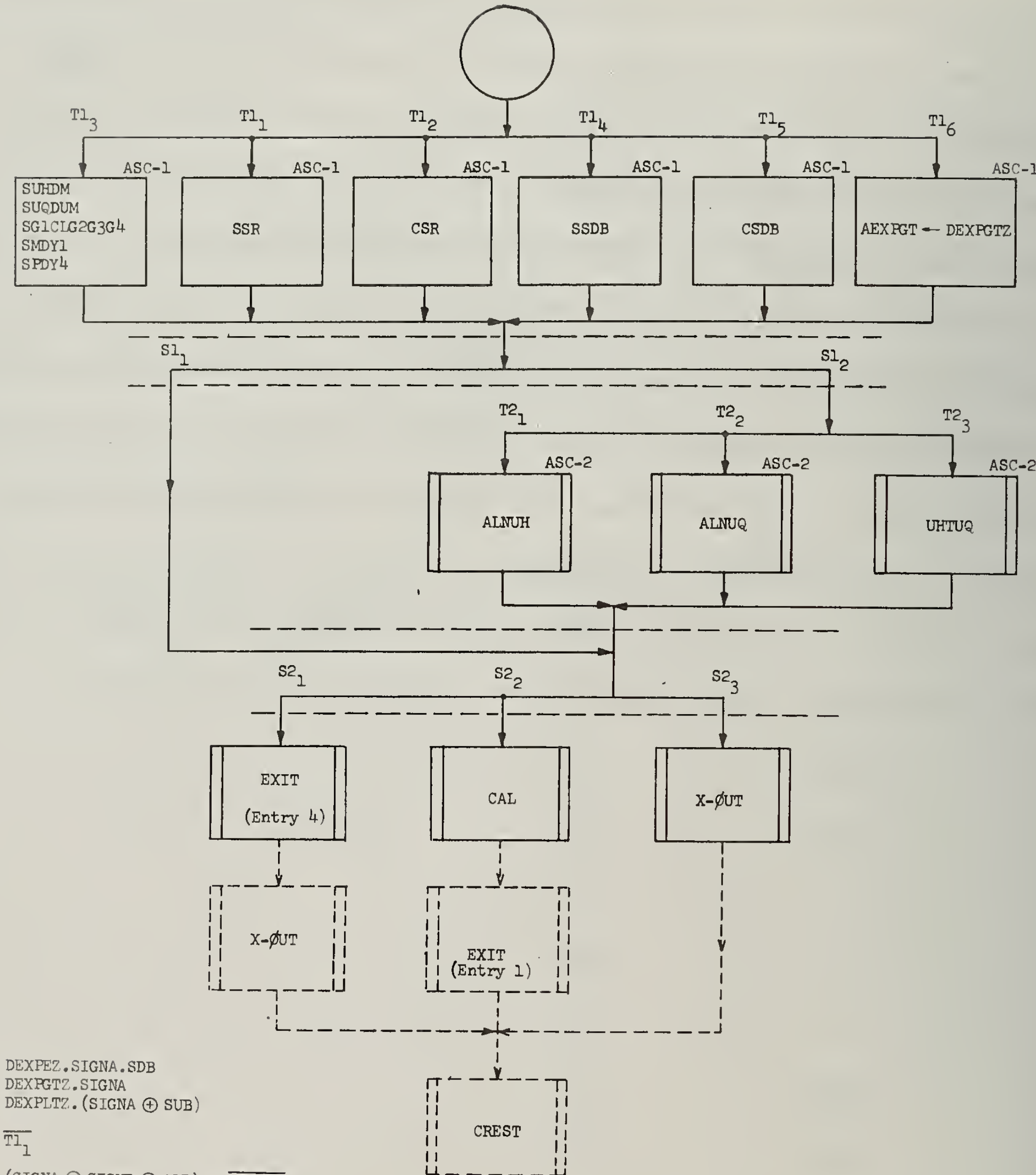
3.5.2.1 OPACAS

Figure 3.5.2.1 shows the control-point flow chart for OPACAS and logic drawing AUO-07-352-01 shows the corresponding logic implementation. Task stage ASC-1 appropriately sets the flip-flops SR (sign of the result) and SDB (sign-digit-bypass) depending on whether the sign of the result is predictable simply by looking at the difference of the exponent and sign of the two operands as well as the arithmetic instruction. A table for the prediction of SR is given below.

Difference of Exponent	SIGNA = SIGNB	Arithmetic order	SDB	SR
> 0	Yes or No	Any	1	SIGNA
< 0	Yes or No	Any	1	SIGNB
= 0	Yes	ADD	1	SIGNA
		SUB	0	0
		CPRA	0	0
	No	ADD	0	0
		SUB	1	SIGNA
		CPRA	1	SIGNA

In addition, this control point (task stage) sets up the path, under task condition Tl_3, for transfer of operands to M and UM registers. It also sets the proper path through the SDS array to be used in Subsequence CAL for calculating the sum or difference of the operands. Besides a control flip-flop AEXPGT is set if the difference of exponents is greater than zero. Now, if the arithmetic

order is either ADD or SUB, the second task stage ASC-2 is entered which calls the operand alignment (shifting) control subsequence (ALNUH or ALNUQ) if the difference of exponents is in range. However, if the exponent unit adder underflows or if the difference of exponent is < -13, it implies that operand A is insignificant compared to operand B and the result is operand B which should be transferred to result register UQ. ASC-2 achieves this by calling upon the subsequence UHTUQ. If the arithmetic order is CPRA, (floating point comparison), the control sequence bypass ASC-2 and proceeds directly to sequence stage S2 where the control branches to any one of the three control subsequences. If there is no overflow/underflow and the difference of exponents is in range (i.e. $< |13|$) or if the sign of the result is unpredictable (SDB = 0), the control branches to subsequence CAL to calculate sum or difference.



$\mathrm{T1_1} = \mathrm{DEXPEZ.SIGNA.SDB}$
$\quad \mathrm{v\ DEXPGTZ.SIGNA}$
$\quad \mathrm{v\ DEXPLTZ.(SIGNA} \oplus \mathrm{SUB)}$

$\mathrm{T1_2} = \overline{\mathrm{T1_1}}$

$\mathrm{T1_4} = (\mathrm{SIGNA} \oplus \mathrm{SIGNB} \oplus \mathrm{ADD})\ \mathrm{v}\ \overline{\mathrm{DEXPEZ}}$

$\mathrm{T1_5} = \overline{\mathrm{T1_4}}$

$\mathrm{T1_6} =$ (NONE)

$\mathrm{T1_3} = \overline{(\mathrm{DEXPLTM13\ v\ EUAUN})}$

$\mathrm{S1_1} = \overline{\mathrm{S1_2}}$

$\mathrm{S1_2} = \overline{\mathrm{CPRA}} \cdot \overline{\mathrm{DEXPEZ}} \cdot \overline{\beta}$

$\mathrm{T2_1} = \overline{\mathrm{CPRA}}\mathrm{.DEXPIPR}$

$\mathrm{T2_2} = \overline{\mathrm{CPRA}}\mathrm{.DEXPINR}$

$\mathrm{T2_3} = \overline{\mathrm{CPRA}}.\overline{\mathrm{T1_3}}$

$\mathrm{S2_1} = \mathrm{CPRA.SDB}$

$\mathrm{S2_2} = (\overline{\mathrm{CPRA}}\ \mathrm{v}\ \overline{\mathrm{SDB}})\ \overline{(\beta\ \mathrm{v}\ \overline{\mathrm{T1_3}})}$

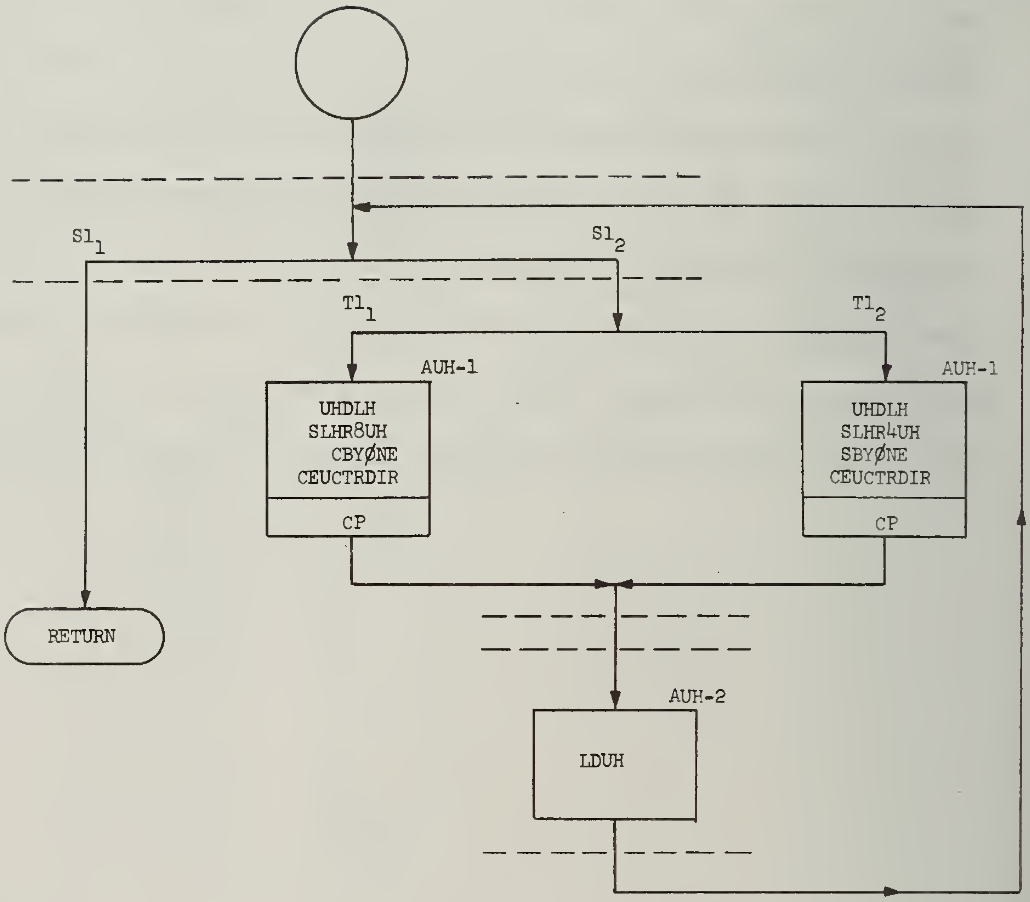
$\mathrm{S2_3} = \overline{\mathrm{CPRA}}.(\beta\ \mathrm{v}\ \overline{\mathrm{T1_3}})$

$\beta = (\mathrm{DEXPGT13\ v\ EUA\emptyset V})$

Figure 3.5.2.1. ASC_OPACAS--Add, Subtract, Compare Algebraic-Operand Alignment Call Set-Up

3.5.2.1.1 ALNUH

Figure 3.5.2.1.1 shows the control point flow chart for ALNUH and logic drawing AUO-07-352-02 shows the corresponding logic implementation. The task stage AUH-1 shifts right the contents of UH register by either 8 or 4 bits and correspondingly decreases the exponent unit counter till the counter contents go to zero. At this instant, the operands are aligned and the subsequence replies back to the calling control point of calling sequence OPACAS. Note that the exponent unit counter and condition (DEXPEZ) detect logic should be completely stabalized before the control loops back again to determine whether to go through another pass of the loop. That is why, it was decided to use second task stage AUH-2 so that enough time is available for condition to stabilize.



Sl_1 = DEXPEZ = EULEZ

$Sl_2 = \overline{Sl_1}$

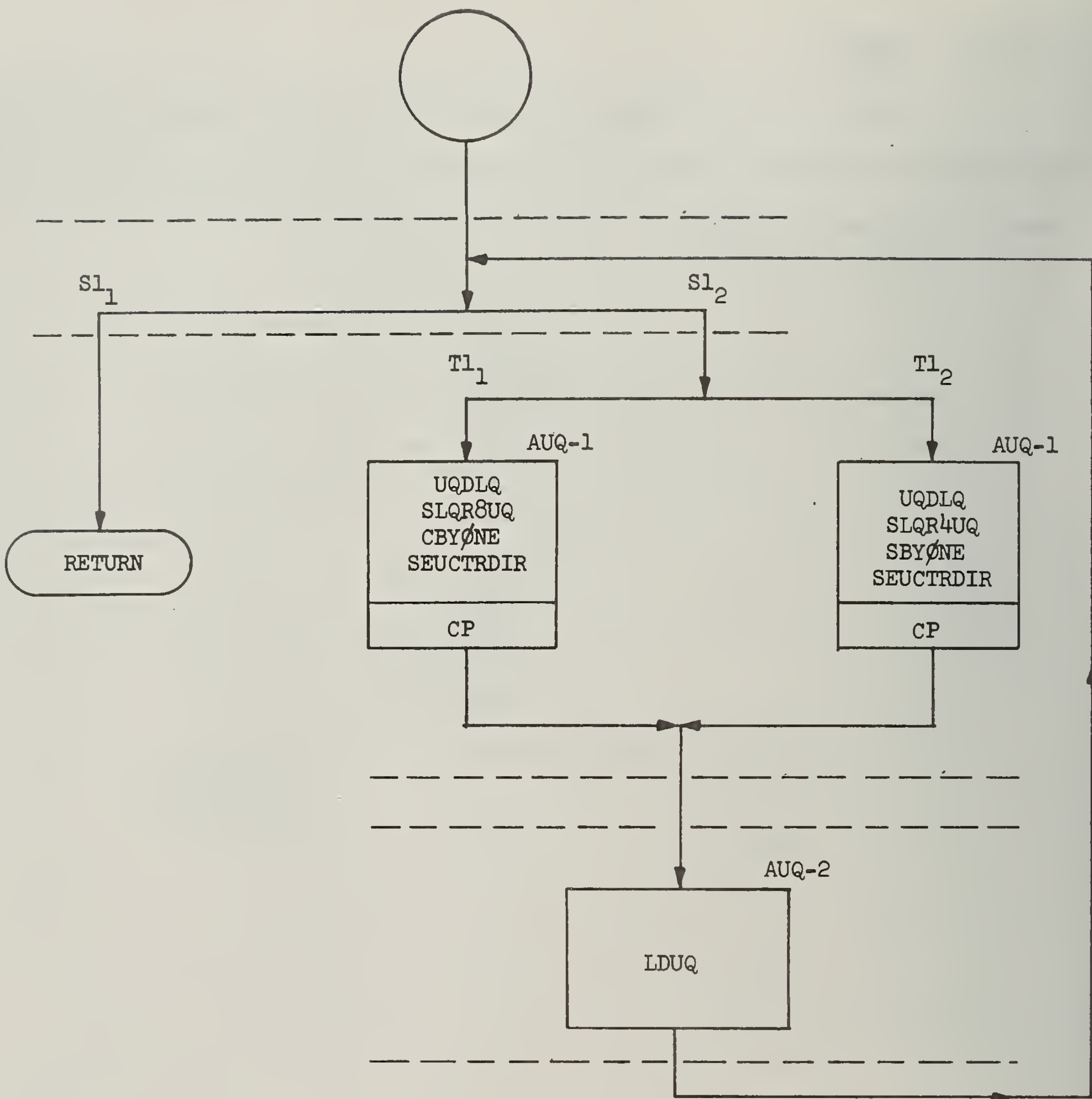
Tl_1 = DEXPGE2

$Tl_2 = \overline{\text{DEXPGE2}}.\overline{\text{DEXPEZ}} = \overline{\text{DEXPGE2}}.\overline{\text{EULEZ}}$

Figure 3.5.2.1.1. ALNUH—Align Contents of Register UH

3.5.2.1.2 ALNUQ

Figure 3.5.2.1.2 shows the control point flow chart for ALNUQ and logic Drawing AUO-07-352-02 shows the corresponding logic implementation. ALNUQ is exactly identical to ALNUH.



Sl_1 = DEXPEZ = EULEZ

Sl_2 = $\overline{\text{DEXPEZ}}$ = $\overline{\text{EULEZ}}$

Tl_1 = DEXPLEM2

Tl_2 = $\overline{\text{DEXPLEM2}}.\overline{\text{DEXPEZ}}$

Figure 3.5.2.1.2. ALNUQ—Align Contents of Register UQ

3.5.2.1.3 UHTUQ

Figure 3.5.1.1.3 and logic Drawing AUO-07-352-02 show the control point flow chart and logic implementation for UHTUQ. Since there is no direct path between registers UH and UQ, the control makes use of the SDS-array to do the transfer. Task stage UHQ-1 sets up the path through the array and also transfers UH to register M. Note that since the transfer through the array begins after the register M has been loaded, the duration of the second delay in the delayed task generation (LDM) should be sufficient not only to load the register M but should be large enough for the new contents of M, travel through the last stage of SDS-array (SMDY4). Now the control enters the second task stage UHQ-2 where the output of SDS-array is transferred to register LM and thence to register UQ by control signal LDUQ. Note that the path between LM and UQ had been established earlier in task stage UHQ-1 by control signal SLMDUQ.

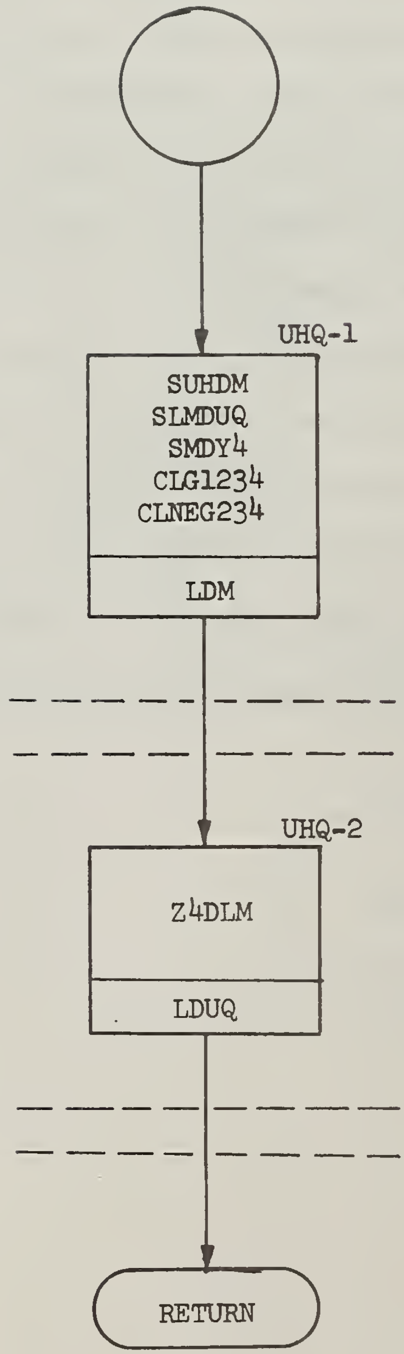


Figure 3.5.2.1.3. UHTUQ—Transfer Contents of Register UH to Register UQ

3.5.2.2 CAL

This Procedure (control subsequence) is used to claculate the sum or difference of the two operands. Two important things to note here are the derivation of Boolean expression (shown in Figure 3.5.2.2.1) for setting and resetting the NEGØ and NEG1 control signals of the SDS-array and the assimilation of the result into conventional number representation. Tables 3.5.2.2.1 and 3.5.2.2.2 show the value of control signals NEGØ and NEG1 for all combinations of SIGNA and SIGNB and for SUB, CPRA and ADD. Since the floating point representation is sign and magnitude representation and maintissa is always positive, the choice of NEGØ and NEG1 should be such that the result of adding or subtracting the mantissas of the two operands in the SDS-array should be positive. Under the column heading "Effective Operation" in the tables, the entries are written so that the mantissa of the result will always be positive in those cases when the sign of the result is predictable. In the cases, where the sign of the result is not predictable a priori, operand B is subtracted from Operand A and if the result is negative, the order of subtraction of the two mantissas is changed by complementing NEG1 as in task stage CAL-3. Note that in the SDS-array, NEGØ negates the mantissa A and NEG1 negates the difference of mantissas A and B. Under the column heading, "Comment", the entries show how the values of NEGØ and NEG1 achieve a positive mantissa of the result.

In the simulation flow chart, an explicit call to subroutine "ASIM" is shown for assimilation of signed digit result into conventional form but the control point flow chart does not show the call explicitly, because some of the necessary control signals for Assimilation (SG1CLG2G3G4, SPDY4) have been set

earlier in task stage ASC-1 in control subsequence OPACAS and others have been set either in control subsequence CREST (e.g. CY2SFF, CY3SFF, CLNEG234) or in task stage CAL-1 (e.g. SUMDXI, SUSDS1, NEGØ, NEG1) and task stage CAL-4 (e.g. Z4DLM).

Task stage CAL-1 sets the control signals NEGØ, NEG1 to appropriate state and also transfers the exponent of the either operand whichever exponent is higher, as the exponent of the result to the register counter EUL in the exponent arithmetic section of the processing hardware.

Task stage CAL-2 is only a dummy stage and provides enough time for SDS-array and propagation logic to settle down. Sequence stage S2 checks whether the sign of the result was predictable a priori; if not, it goes to task stage CAL-3 to check whether the sign of the difference/sum of the two mantissas at the output of SDS-array is positive or negative. If it is negative as determined by task condition $T3_1$, the order of the subtraction is changed. In this context, the difference from the simulation flow chart should be carefully noted. In the simulation flow chart, Z_7 is examined to see if the assimilated output of Z_4 is negative, whereas in control hardware, $Z_{4,1}$ i.e. the leftmost bit of Z_4 output is used. It does not really make any difference because $Z_{4,1}$ and $Z_{4,7}$ are the same always.

Task stage CAL-4 transfers the assimilated result's mantissa to register UQ via register LM and the control then branches to subsequence EXIT.

Function: SUB, CPRA.

SIGNA	SIGNB	DEXP	SR	Effective Operation	SDB	NEGØ	NEG1	Comment
0	0	>0	0	A-B	1	0	0	A-B
		< 0	1	-(B-A)	1	0	1	-(A-B)
		=0	?*	A-B	0	0	0	(A-B)
1	1	>0	1	-A+B =-(-B+A)	1	0	0	(A-B)
		< 0	0	-A+B	1	0	1	-(A-B)
		=0	?*	-A+B	0	0	1	-(A-B)
0	1	>0	0	A+B	1	1	1	-(-A-B)
		< 0	0	A+B	1	1	1	-(-A-B)
		=0	0	A+B	1	1	1	-(-A-B)
1	0	>0	1	-(A+B)	1	1	1	-(-A+B)
		< 0	1	-(A+B)	1	1	1	-(-A-B)
		=0	1	-(A+B)	1	1	1	-(-A-B)

Table 3.5.2.2.1 - Determination of Values of NEGØ , NEG1 for arithmetic orders SUB and CPRA

Function: ADD

SIGNA	SIGNB	DEXP	SR	Effective Operation	SDB	NEGØ	NEG1	Comment Magnitud
0	0	>0	0	A+B	1	1	1	-(-A-B)
		< 0	0	A+B	1	1	1	-(-A-B)
		=0	0	A+B	1	1	1	-(-A-B)
1	1	>0	1	-A-B	1	1	1	-(-A-B)
		< 0	1	-A-B	1	1	1	-(-A-B)
		=0	1	-A-B	1	1	1	-(-A-B)
0	1	>0	0	A-B	1	0	0	(A-B)
		< 0	1	(A-B) =-(B-A)	1	0	1	-(A-B)
		=0	?*	A-B	0	0	0	(A-B)
1	0	>0	1	-A+B =-(A-B)	1	0	0	(A-B)
		< 0	0	-A+B =(B+A)	1	0	1	-(A+B)
		=0	?*	-A+B	0	0	1	-(A-B)

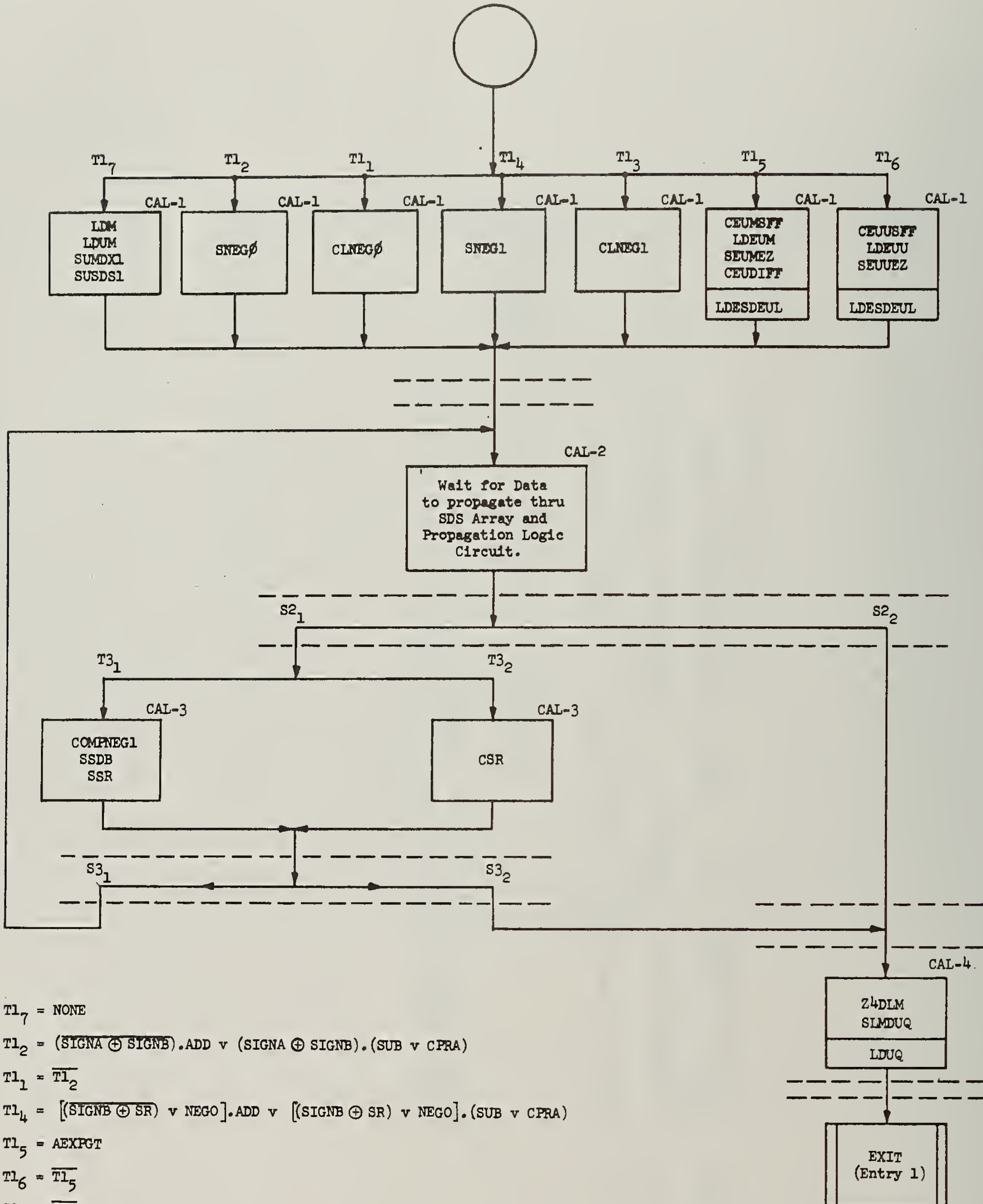
Table 3.5.2.2.2 - Determination of Values of NEGØ, NEG1 for arithmetic order ADD.

$$\mathrm{NEG\emptyset} = (\overline{\mathrm{SIGNA} \oplus \mathrm{SIGNB}})\cdot\mathrm{ADD} \vee (\mathrm{SIGNA} \oplus \mathrm{SIGNB})\cdot(\mathrm{SUB} \vee \mathrm{CPRA})$$

$$\mathrm{NEG1} = ((\overline{\mathrm{SIGNB} \oplus \mathrm{SR}}) \vee \mathrm{NEG\emptyset})\cdot\mathrm{ADD} \vee ((\mathrm{SIGNB} \oplus \mathrm{SR}) \vee \mathrm{NEG\emptyset})\cdot(\mathrm{SUB} \vee \mathrm{CPRA})$$

$$\mathrm{SR} = \mathrm{DEXPEZ}\cdot\mathrm{SIGNA}\cdot\mathrm{SDB} \vee \mathrm{DEXPGTZ}\cdot\mathrm{SIGNA} \vee \mathrm{DEXPLTZ}\cdot(\mathrm{SIGNA} \oplus \mathrm{SUB})$$

Figure 3.5.2.2.1 - Boolean Expressions for Control Signals NEGØ, NEG1 and SR, for a priori predictability of sign of Result.



$T1_7$ = NONE

$T1_2 = (\overline{SIGNA \oplus SIGNB}).ADD \vee (SIGNA \oplus SIGNB).(SUB \vee CPRA)$

$T1_1 = \overline{T1_2}$

$T1_4 = [(\overline{SIGNB \oplus SR}) \vee NEGO].ADD \vee [(SIGNB \oplus SR) \vee NEGO].(SUB \vee CPRA)$

$T1_5$ = AEXPGT

$T1_6 = \overline{T1_5}$

$S2_1 = \overline{SDB}$

$S2_2$ = SDB

$T3_1 = Z_{4,1}$

$T3_2 = \overline{T3_1}$

$S3_1$ = SR

$S3_2 = \overline{S3_1}$

Figure 3.5.2.2. CAL--Sum or Difference Calculation

3.5.2.3. ASIM

Figure 3.5.2.3 shows the control point flow chart for the subroutine control sequence ASIM. This control sequence is used to convert the signed digit numbers into conventional binary representation. This is a very common and often used control subsequence of the AU control. This control point flow chart very closely resembles the simulation flow chart ASIM.

This control subsequence consists of only two task stages. Task stage ASM-1 sets up the SDS-array such that the input Y2 and Y3 to the SDS2 and SDS3 respectively are all zero. Also the gate signals Gi and negate signal NEGi to SDS2, 3, 4 are also set to state 'zero'. Input Y4 to SDS4 comes from the output of propagation logic and signals USDS1 and UMDX1 transfer the contents of registers US and UM to the SDS-array. Control signals for SDS1 are set to proper state before calling the subroutine control subsequence ASIM.

After the data to be assimilated has filtered through the SDS-array and propagation logic, the assimilated data is available at the output of SDS4 and then task stage ASM-2 provides the gate control signal Z4DLM which loads the assimilated data in register LM.

After this, the control returns back to the calling control point which called this subsequence.

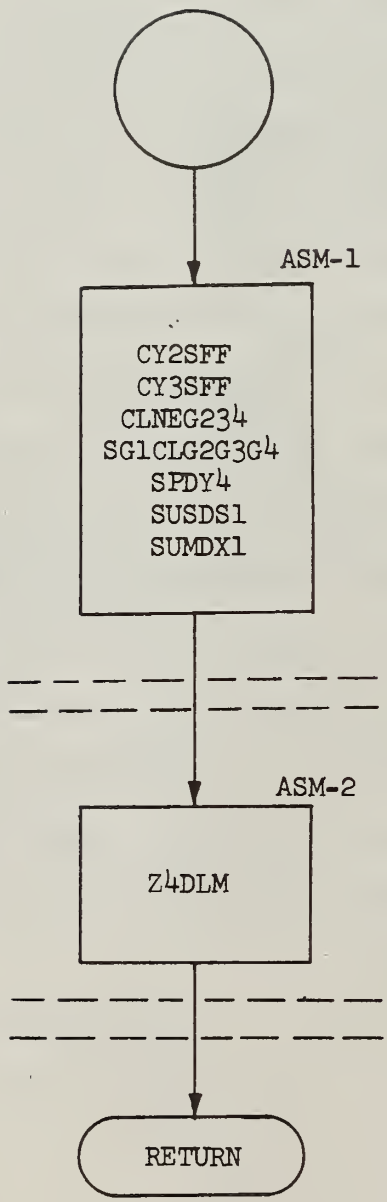


Figure 3.5.2.3. ASIM—Conversion from Signed-Digit to Conventional Representation

3.5.3 Logic Implementation

Logic implementation of the various control subsequences of ASC control sequence is summarized in the table below which shows the correspondence between control point flow charts and their logic implementation. Besides, the table also shows the drawing number of any task signal collector (TSC) drawings for the sequence.

Control Sub-sequence Name	CP flow chart Figure Number	Corres. Logic Drawing No.
OPACAS	3.5.2.1	AUO-07-352-01
ALNUH	3.5.2.1.1	-352-02
ALNUQ	3.5.2.1.2	-352-02
UHTUQ	3.5.2.1.3	-352-02
CAL	3.5.2.2	-352-03
ASIM	3.5.2.3	-371-04
REFOTRAN		-332-05 -352-04, 05, 06
Task Signal Collector "A_1"		-352-07
Task Signal Collector "A_2"		-352-08

3.6 MULTIPLY (MPY) INSTRUCTION CONTROL SEQUENCE

This control sequence is used for the execution of Multiplication process for both floating and fixed point number systems.

3.6.1 Global Flow Description

Figure 3.6.1.1 shows the global flow diagram for the MPY control sequence. It is mainly comprised of two subsequences MPY-REPROG and MPYEND besides the subsequence REFOTRAN which is common to all the arithmetic orders. Control subsequence MPY-REPROG provides control signals for the multiplication of the two operands. The two operands in registers UQ and UH are in conventional form but the product formed by the subsequence MPY-REPROG is in the signed digit form.

Since the result to be transmitted to the TP should be in conventional form, the subsequence MPYEND calls upon the assimilation subsequence ASIM to convert the product to the conventional form. Besides, the subsequence MPYEND checks whether, in case of fixed point multiplication, the product has overflowed and accordingly sets the overflow flip-flop ϕV to the appropriate state.

At the end of these two subsequences, the control branches to subsequence REFOTRAN which formats the result properly and then transmits the formatted result along with proper indicators and flags to the waiting TP via the exchange net.

Finally the control goes to subsequence CREST which initializes and readies the control for another arithmetic order.

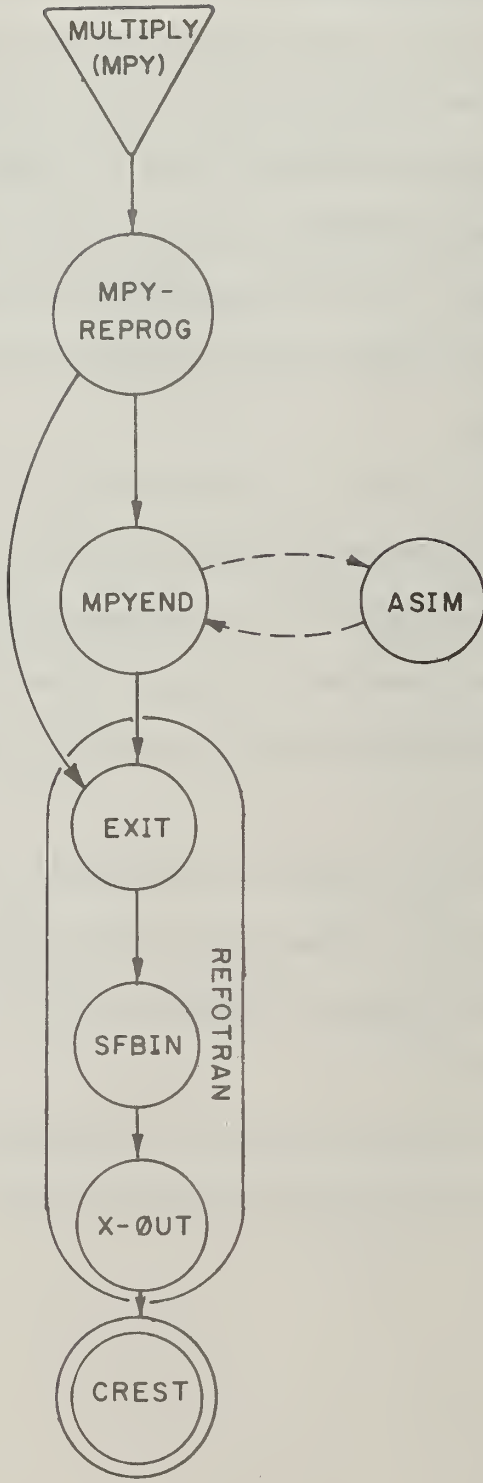


Figure 3.6.1.1. Global Flow Diagram for "MPY--Multiplication Process (Fx. Pt. and Fl. Pt.)" Control Sequence

3.6.2 Control-Point Flow Description

Control point flow charts are almost identical to the simulation flow charts with a few exceptions. For example, in the simulation flow chart, the setting up of the SDS-array and M-shift array is done sequentially, one SDS-array and M-shift array stage at a time whereas in actual control, this is done in parallel by the already wired in combination logic and task stage MPY-3. Another exception is the use of only one counter ICC and some combinational logic for 'multiplication-loop-termination' condition detect, whereas the simulation flow chart shows the use of the register NCC and the counter ICC for the same purpose.

A brief description of the action performed by each task stage is given below.

Multiplicand and the multiplier are in registers UH and UQ respectively, being placed there by the VIN control sequence. In addition, the exponents of the two operands are processed in the exponent arithmetic unit, where the two exponents are added and exponent overflow/underflow condition is detected. In sequence stage stage SO, the sequence condition SO_1 determines whether operand is zero and if so, task stage MPY-2 sets the result register UQ to all zero and also sets the register EUL, which holds the exponent of the result, to all zero state. Then the control sequence branches to subsequence EXIT where flags are set and eventually the result is transmitted to the TP as explained earlier in sequence REFOTRAN.

If the number type of the operand is fixed point, then the contents of the register UH(i.e. the multiplicand operand) is shifted right by 8 bits in task stage MPY-1 so that the same logic and control sequence may be utilized for both the fixed point and floating point operands' multiplication.

In case of floating point non-zero operands (condition SO_2), the task stage MPY-3 sets up the data path for the transfer of the multiplicand from the UH register to register M and then provides the gate signal LDM for the actual transfer. Task stage MPY-3 also sets up the SDS-array for the multiplication loop. Besides it clears the loop counter ICC which is bidirectional and sets it up for the count-up state. Finally the data path between the register UQ and the multiplier recoder (SUQDMR) is set up. The output of the multiplier is directly connected to the M-shift array which shifts the multiplicand in the M-register appropriately into the SDS-array.

Task stage MPY-4 updates the counter ICC and transfers UQ contents direct to LQ for the right shifting of the multiplier. The path for right shift from LQ to UQ was set up in task stage MPY-3 (SLQR8UQ). This task stage also provides the necessary time interval for the SDS-array to settle down.

Task stage MPY-5 transfers the partial product at the output of SDS-array to the registers LM and LS and also gates the output of LQ to UQ with an eight-bit right shift. Since the task stage MPY-6 provides the gating signals for the transfer of the contents of LS and LM (which is the partial product) into registers US and UM respectively, a special action has to be taken, in case long fixed point product generation is complete, before entering the task stage MPY-6 after going around the multiplication loop four times. In this case, the task state MPY-5 sets up a data path for direct transfer of redundant-form product into registers US, UM for conversion to conventional form by subsequence ASIM, later on, in the sequence. This special action is necessary because task stage MPY-3 had set up a path between LS, LM and US, UM which would have right shifted the contents of LS, LM by 8 bits.

Task stage MPY-6 besides gating the contents of LS, LM into US, UM with an appropriate shift, also generates the control signal LDMEXPRE which provides, in case of floating point, extended precision of four bits. For a detailed discussion of extended precision, refer to Section 2.9.3 of DCS Report No. 366. Care should be taken to understand the timing relationship between the control signals MEPGATE and MEPLATCH because of the Earl latch type buffer used for extended precision bits MEPPO, $MEPB_1$ - $MEPB_4$.

Sequence stage S6 detects whether the redundant form product generation loop should be terminated or continued. The termination condition is detected by the logic for control signal MPYSTOP whose Boolean expression is given below.

$$\mathrm{MPYSTOP} = \mathrm{FLT}\cdot\mathrm{ICCEQ7} \vee \mathrm{LFIX}\cdot\mathrm{ICCEQ4} \vee \mathrm{SFIX}\cdot\mathrm{ICCEQ2}.$$

Task stage MPY-7 breaks the data path between UQ and the multiplier recoder and sets up the control signals NEGO, NEG1 and data input Y1 to appropriate state for the assimilation of the redundant form product to conventional form.

Task stage MPY-8 calls upon the subsequence ASIM which provides the proper signals for converting the result to the conventional form. The conventional form result is available in register LM.

Task stage MPY-9 transfers the result to result register UQ.

Sequence stage S9 branches the control to task stage MPY-10 if the multiplication operands are fixed point type. Task stage MPY-10 checks for the overflow condition and accordingly sets the overflow flip-flop ϕV to the appropriate state. In case of floating point operands, the control branches to sequence REFOTRAN via subsequence EXIT directly from sequence stage S9 whereas in case of fixed point operands, the control goes to terminating sequence REFOTRAN at the end of task stage MPY-10.

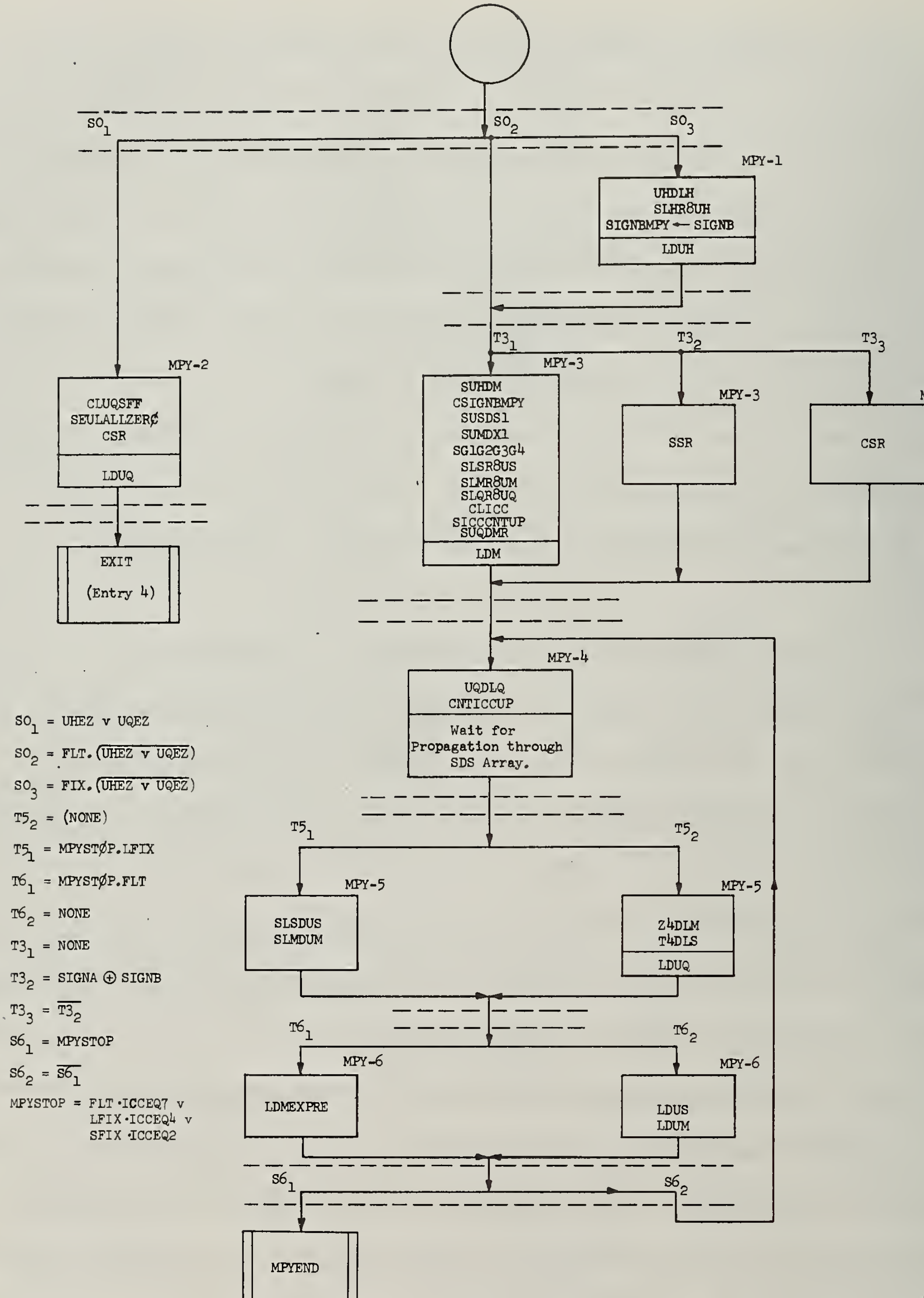
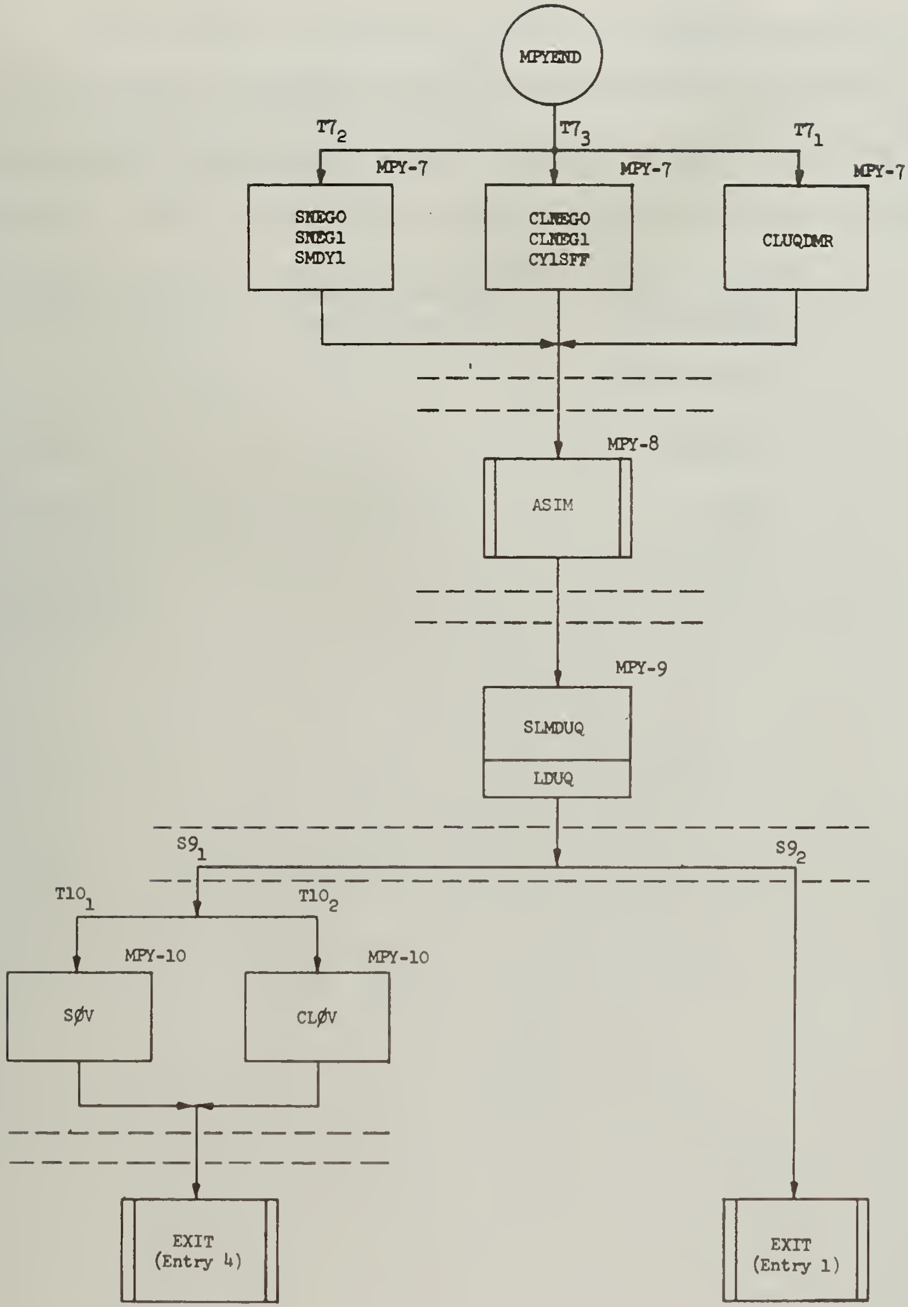


Figure 3.6.2.1. MPY_REPROG—Multiply (Fl. Pt. and Fx. Pt.)—
Redundant Form Product Generation



$T7_2$ = FLT.UQ_{65}

$T7_3$ = $\overline{T7_2}$

$T7_1$ = NONE

$S9_1$ = FIX

$S9_2$ = FLT

$T10_1$ = SFIX.SR.$\overline{\text{UQ45AØ}}$
v SFIX.$\overline{\text{SR}}$.$\overline{\text{(UQ4EZ.UQ5EZ)}}$
v $\overline{\text{SFIX}}$.SR.$\overline{\text{UQ03AØ}}$
v $\overline{\text{SFIX}}$.SR.$\overline{\text{UQ03EZ}}$

$T10_2$ = $\overline{T10_1}$

Figure 3.6.2.2. MPYEND—Conventional Form Product Formation and ØV Status Indicator Set-Up

3.6.3 Logic Implementation

Logic implementation of the various control subsequences of MPY control sequence, other than those of terminal sequence REFOTRAN, is summarized in the table below.

Control Subsequence Name	CP flow chart Figure Number	Corres. logic Drawing No.
MPY-REPROG	3.6.2.1	AUO-07-361-01
MPYEND	3.6.2.2	-361-02
ASIM	3.5.2.3	-371-04
REFOTRAN		-331-05 -352-04, 05, 06

3.7 DIVIDE INSTRUCTION CONTROL SEQUENCE

This control sequence is entered whenever the arithmetic order to be processed is 'DIVISION instruction' for both the floating point and fixed point operands. This is the longest sequence among all the arithmetic orders processed by the AU and fixed point division forms the major part of the control sequence.

3.7.1 Global Flow Description

Figure 3.7.1.1 shows the global flow diagram of 'Divide' instruction control sequence for both the fixed and floating point operands. It consists of many control subsequences (procedures) and a brief description of each is given below.

The very first procedure subsequence to be entered is DFL which, in conjunction with subroutine control subsequence DIVIDE, provides for the division of floating point operands. It also sets up the data paths for the transfer of operands to the proper registers M and UM besides setting the loop counter and the 'sign of the result' flip-flop SR. This initial set-up operation is common to both fixed and floating point operands and hence forms the initial part of DFL subsequence. This subsequence also provides, after the initial set up, an exit to subsequence DFX-DIZETCOM which forms the initial part of sequence DFX used for 'fixed point division' - arithmetic order. The subsequence DFL ultimately branches to EXIT subsequence of terminal sequence REFOTRAN where the result is formatted and then transmitted to the processor.

In case of fixed point division, the first subsequence is DFX-DIZETCOM, except for the initial entrance to DFL as explained earlier. In this subsequence, a test is made to see if either the dividend or the divisor is zero. If the divisor is zero, an error will occur. The ϕV will be set and the bogus result is $2^{31}-1$. For a zero dividend and nonzero divisor, the quotient is zero, result register UQ is cleared and then control branches to EXIT subsequence of terminal sequence REFOTRAN. For nonzero dividend and divisor, a test is made if any or both of the operands is negative. If so, the contents of register UQ which holds both the operands are 2's complemented so that the quotient is

positive - and same control logic can be shared, as is used with floating point operands where the mantissas are always positive. The control then branches to another procedure subsequence SCALFIXDR.

In this subsequence SCALFIXDR, both the divisor and the dividend in registers UH and UQ respectively are scaled by left shifting UH and UQ till the divisor is $\geqslant 1/2$ and the dividend is $\geqslant \frac{1}{2^8}$. At the same time, a record is kept in control counters NDRL8, NDDL8, NDRL1 and control flip-flop NDRL4 of the total number of shifts made, so that both the quotient and the remainder can be post-scaled properly to calculate the correct result. This scaling is necessary so that the same algorithm and hence the control logic can be shared as with floating point operands.

Now that the dividend and divisor are properly scaled, the control passes to subsequence FORMQAREM where the quotient and the remainder are calculated by calling on the subroutine control sequence DIVIDE. Actual division is performed by the subroutine sequence DIVIDE. The quotient and remainder obtained from the control sequence DIVIDE is in signed-digit form, but task stages DFX-22 to DFX-24 convert these into conventional form.

After the quotient and remainder are assimilated into conventional form, the control goes to procedure control subsequence COSREM. The main function of this subsequence is to make the sign of remainder the same as that of the dividend and sets a control flip-flop NEGR for any necessary correction in the quotient at the same time. After the sign of the remainder is corrected, the remainder is assimilated into the conventional form and the control goes to subsequence QASS, if the arithmetic order being processed is fixed point division. However, if the arithmetic order is CVD (Convert to decimal) which shares the subsequences

FORMQAREM and COSREM with 'Fixed point Division' arithmetic order, the control replies back to the calling point in CVD control sequence.

In the subsequence QASS, the necessary correction of unit indicated by the state of control flip-flop NEGR, is made in the magnitude of the quotient and at the same time, a proper sign is attached to the quotient depending upon whether the sign of the result indicated by SR = (SIGNA $\oplus$ SIGNB) is positive or negative.

Finally, the control passes on to the control subsequence RESCALEREM before entering the terminal sequence REFOTRAN. In the control subsequence RESCALEREM, the assimilated remainder which is in register UQ is scaled again to get the correct remainder. This post scaling is necessary because the divisor was earlier scaled to lie between 1/2 and 1 due to the requirements of 'Model Division'. After post-scaling, the assimilated quotient in register LM is transferred to bytes 4-7 of the result register UQ which already holds the remainder in its bytes 1-3.

Ultimately, the control enters the terminal sequence REFOTRAN where the flags and indicators are set and the result transmitted to the TP, before finally entering the initialization subsequence CREST.

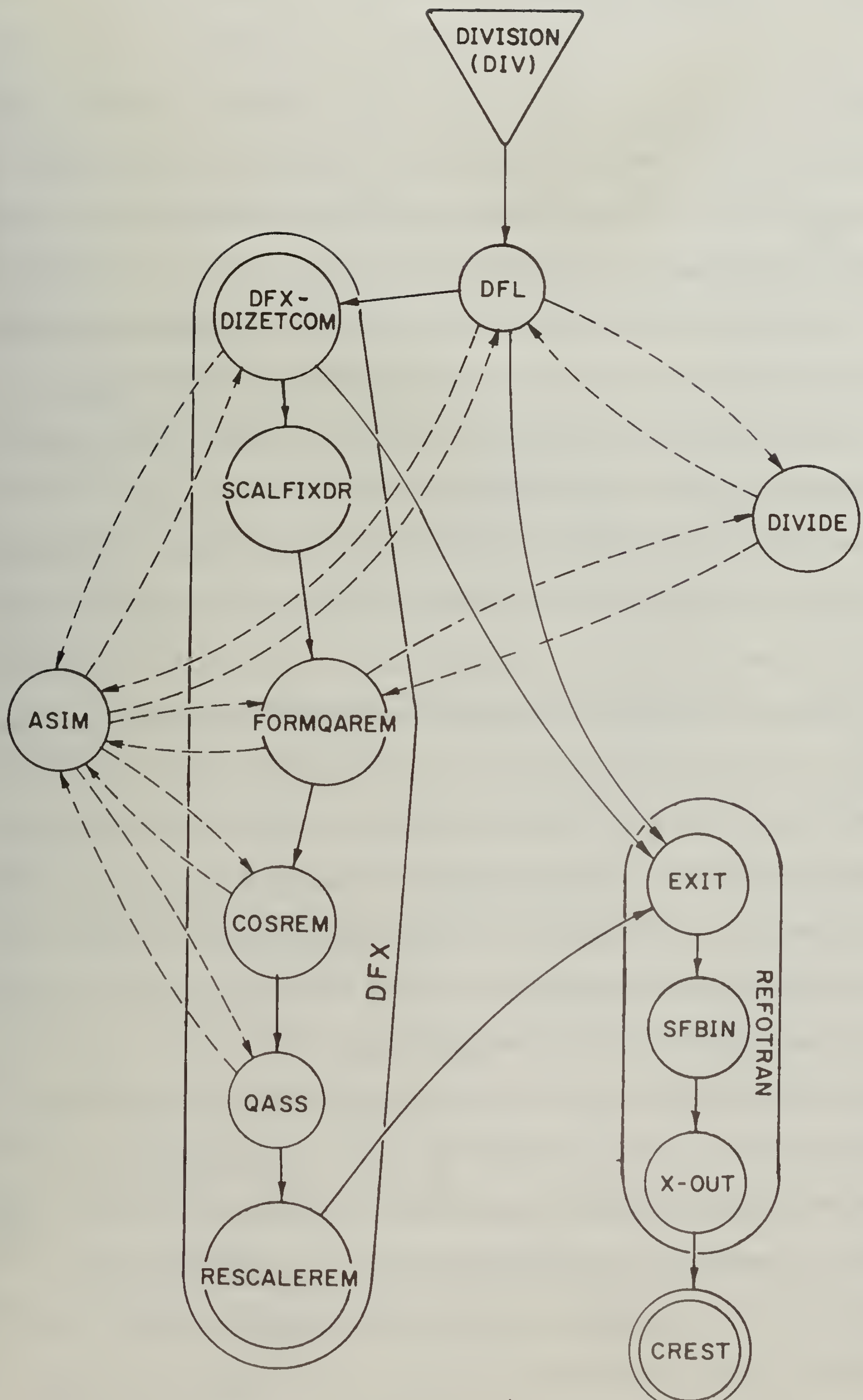


Figure 3.7.1.1. Global Flow Diagram for "DIV--Division Process
(Fx. Pt. and Fl. Pt.)" Control Sequence

3.7.2 Control Point Flow Description

3.7.2.1 DFL

Figure 3.7.2.1 shows the control point flow chart for procedure control subsequence DFL. It resembles very closely the simulation flow chart with one or two differences. Task stage DFL-1 sets up data path for transfer of operands to registers M and UM and at the same time initializes the division loop counter ICC and sets the flip-flop SR to appropriate state. Register M and UM will respectively contain the divisor and the dividend. This state is common to both the fixed point and floating point divide.

Sequence stage S1 checks for either operand being zero. If dividend is zero ($S1_2$), the task stage DFL-2 is entered which sets the exponent of the result to zero, the mantissa in UQ is already zero. If mantissa of the divisor is zero ($S1_3$), control flip-flop ϕV, to indicate overflow, is set in task stage DFL-3 and result register UQ and exponent is set to all '1' state. Both the task stages DFL-2 and DFL-3 lead finally to sequence REFOTRAN via subsequence EXIT. If the arithmetic order being processed is fixed point division, the control exits to sequence DFX. In case of non-zero floating point operands ($S1_4$), the control branches to either task stage DFL-4 or DFL-6, according as the divisor (UH) is in the range $\frac{1}{2} \leq (UH) \leq 1$ or not respectively.

In task stage DFL-4 and DFL-5, the contents of UH and UQ are left shifted till the divisor in UH lies between $\frac{1}{2}$ and 1. A departure from the simulation flow chart logic should be noted here. In the simulation flow chart, only the divisor is scaled, not the dividend and the quotient is scaled in opposite direction by the same amount as the divisor towards the end of the sequence. However, in the control both the operands are scaled at the same

time. At first thought, it may appear to be wrong but the fact that any significant bits of the dividend in UQ go into first byte of UQ and are not lost should convince the reader that this will work. Note that the top eight bits of scaled dividend do take part in the calculation of the quotient.

One more thing to be noted here is that the DFL control assumes that the operands are normalized to base 16 and it is the TP's responsibility to see that it is true. No check is made in AU for this purpose.

The reason to examine UH_9 for scaling of the divisor instead of UH_1 is that bits UH_{1-8} are zero due to the loading of the operands. Further it is necessary that dividend be less than the divisor, and it is ensured by the fact that scaled dividend in UM will contain at least five leading zeros whereas the divisor in M_{9-64} has no leading zero as far as calculation of quotient is concerned.

Task stage DFL-6 gates the scaled operands in registers M and UM and also loads the loop counter ICC.

Task stage DFL-7 sets up a call to subroutine control sequence DIVIDE which provides the control signals to the combinational logic of Model Division and SDS-array to perform the actual division. For details of the subsequence DIVIDE, see Section 3.7.2.2. DIVIDE control subsequence provides the quotient in signed-digit form in the register UH and UQ and task stages DFL-8, 9, 10 assimilate the quotient into conventional binary form. The assimilated quotient is available in register LM from where it is transferred to the result output register UQ by task stage DFL-11. Finally the control enters the terminal sequence REFOTRAN via subsequence EXIT where the result is normalized, properly formatted and then transmitted to the TP.

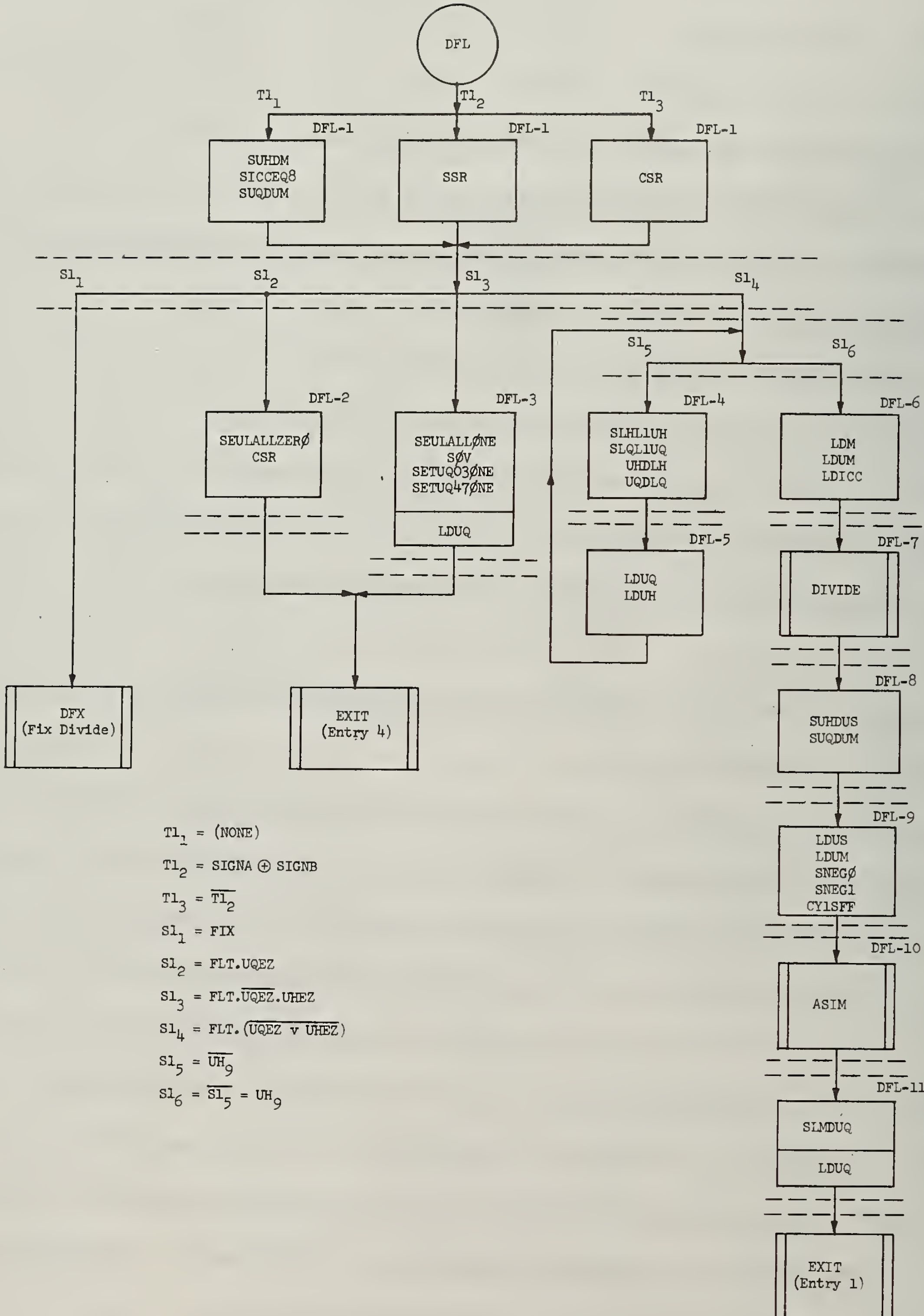


Figure 3.7.2.1. DFL—Floating Point Division

3.7.2.2 DIVIDE

Figures 3.7.2.2.1 and 3.7.2.2.2 shows the control point flow chart for the subroutine control subsequence DIVIDE. This subsequence provides the necessary timing control signals to logic hardware of MODEL DIVISION and SDS-array to perform the actual division and calculate the quotient and/or the remainder. The major difference with the simulation flow chart procedure DIVID lies mainly in the sequential nature of simulation as compared to combinational logic hardware of MODEL DIVISION used to calculate the quotient in actual Control hardware. The simulation flow chart calls upon the procedure for SDS everytime it is needed in contrast to control hardware which makes use of the four copies of SDS physically available in SDS-array.

Task stage DIV-1 initializes the SDS-array to all clear state in order to set the register LM and LS to all zero state - in task stage DIV-2. It is very important that LS and LM be all clear to zero state before actual division and quotient calculation begins. Since this subsequence is also used by sequence CVD where the divisor is always ten, the task stage DIV-2 sets the control signal DRTEN if the arithmetic order is CVD, otherwise it sets the control signal MDDINT. Both the control signals DRTEN and MDDINT are input to "Divisor Interval Select" table-look up logic.

Task stage DIV-3 sets up the SDS-array, the data paths for looping the SDS-array for division loop, and the data paths for left shifting the registers UQ and UH to make space for the quotient. Note that in one pass of the complete SDS-array, 8 bits of the quotient are calculated.

Sequence stage S3 tests for the terminating condition of the division loop. If the necessary passes through the loop are not finished, the task

stage DIV-4 activates the control signal DODMD which transfers most significant six digits of the signed-digit partial remainder (dividend for the very first time) in registers of US and UM to Model Division hardware There, these six digits are assimilated into conventional form and are used in conjunction with 'Divisior Interval Select' logic output to calculate the quotient. This is shown in Figure 3.7.2.2.3 which shows the block diagram of model division. At the same time, contents of UQ and UH are transferred to LQ and LH in order to left shift the contents of UQ and UH later on in task stage DIV-9. This task stage calculates first two bits of the quotient of this pass of the loop (note that each pass obtains 8 bits of quotient). After the output of quotient select table is completely stable, the delayed control task signal LDQMS12 gates the quotient digits into the quotient buffer QM and QS. Since the quotient digits' calculation involve many levels of logic, the gating signal LDQMS12 should be delayed as much as necessary in order that the quotient digits are stable before being gated.

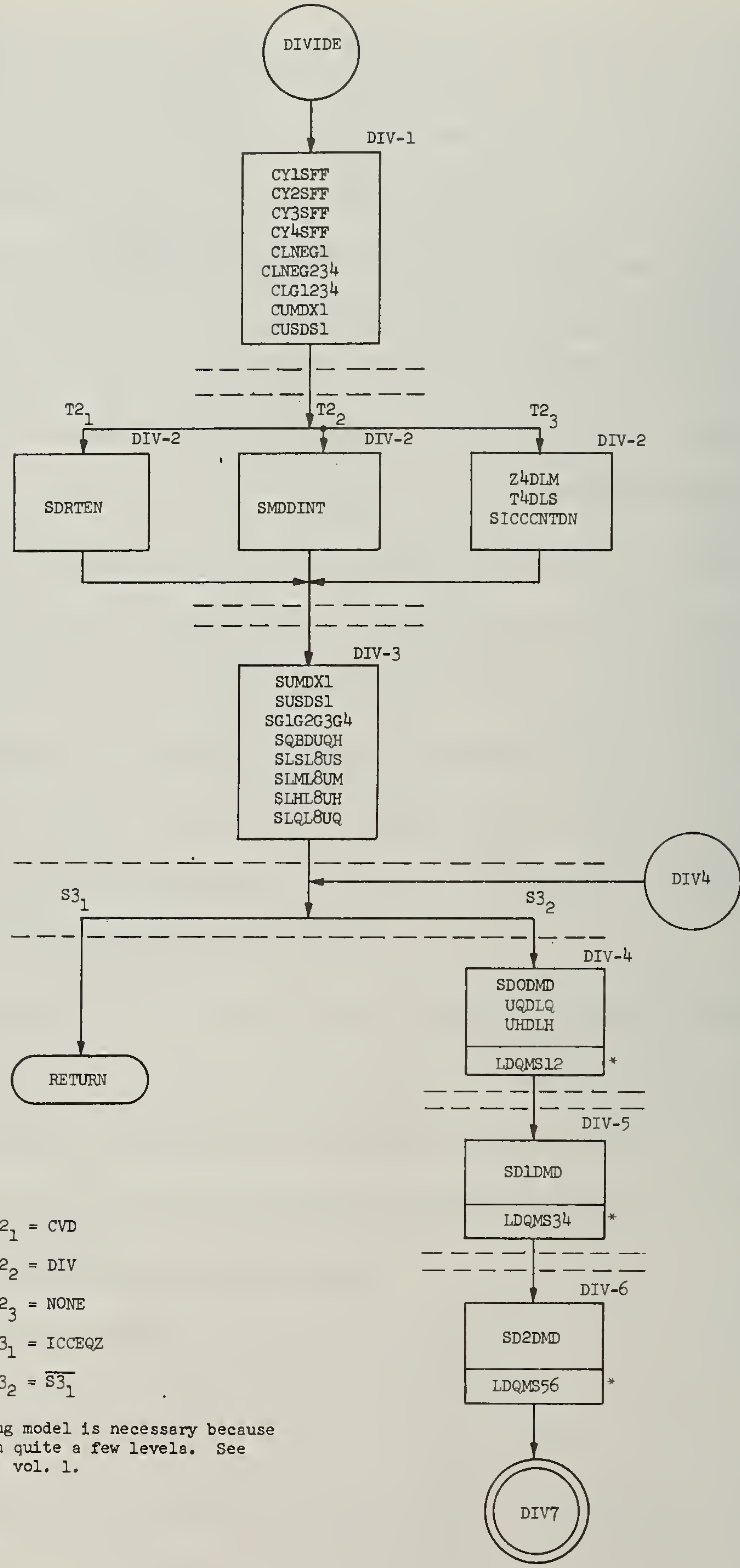
Task stages DIV-5, DIV-6, DIV-7 also calculate the other six digits, 2 digits each, of the quotient in the same way as the task stage DIV-4. In the simulation flow charts, the setting of NFBM is shown explicitly whereas in the actual hardware, this is wired-in already as shown in Drawing AUO-07-210201. It is achieved by the following Boolean logic.

$$PS_1 = (US_1 \oplus LM_8)\ DODMD \vee (T_{1,3} \oplus Z_{1,2})D1DMD \vee$$

$$(T_{2,5} \oplus Z_{2,4})\ D2DMD \vee (T_{3,7} \oplus Z_{3,6})D3DMD.$$

Task stage DIV-8 deactivates the Model Division logic, whereas the task stage DIV-9 and DIV-10 provide the gating signals for the shift of the contents of UQ, UH, and of LS and LM into US and UM. At the same time, the task stage

DIV-9 decrements the loop counter by unity and the control loops back to sequence stage S3. When all the passes through the division loop are complete, the quotient is in registers UQ and UH and the remainder is in LS and LM. The control now returns back to the control point in main sequence where this subroutine control sequence was called.



T2$_1$ = CVD

T2$_2$ = DIV

T2$_3$ = NONE

S3$_1$ = ICCEQZ

S3$_2$ = $\overline{S3_1}$

* Large amount of time in timing model is necessary because SD1DMD signals travel through quite a few levels. See figure 2.10.2.2 of AU manual, vol. 1.

Figure 3.7.2.2.1. DIVIDE—Redundant Form Quotient Generation

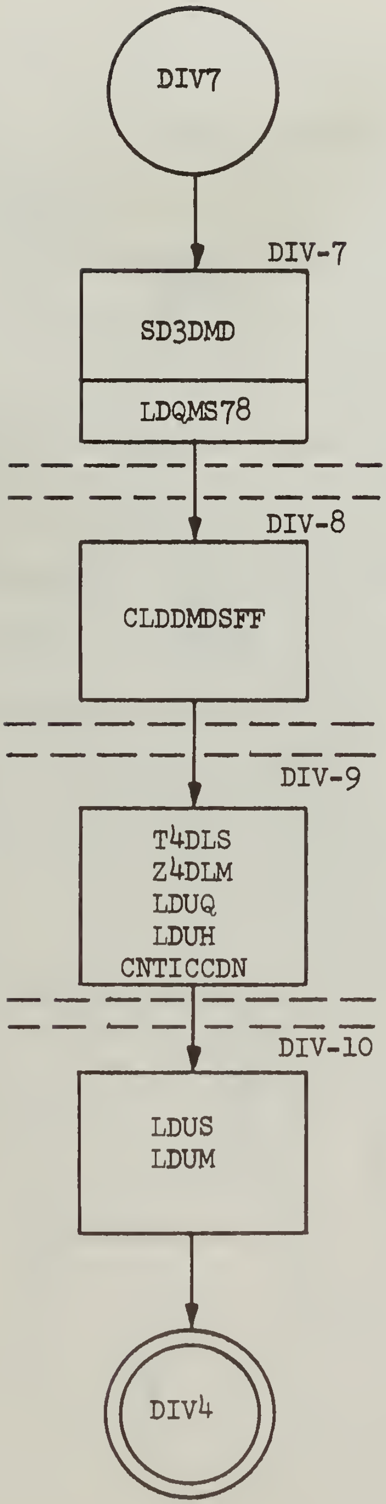


Figure 3.7.2.2.2. DIVIDE—Redundant Form Quotient Generation

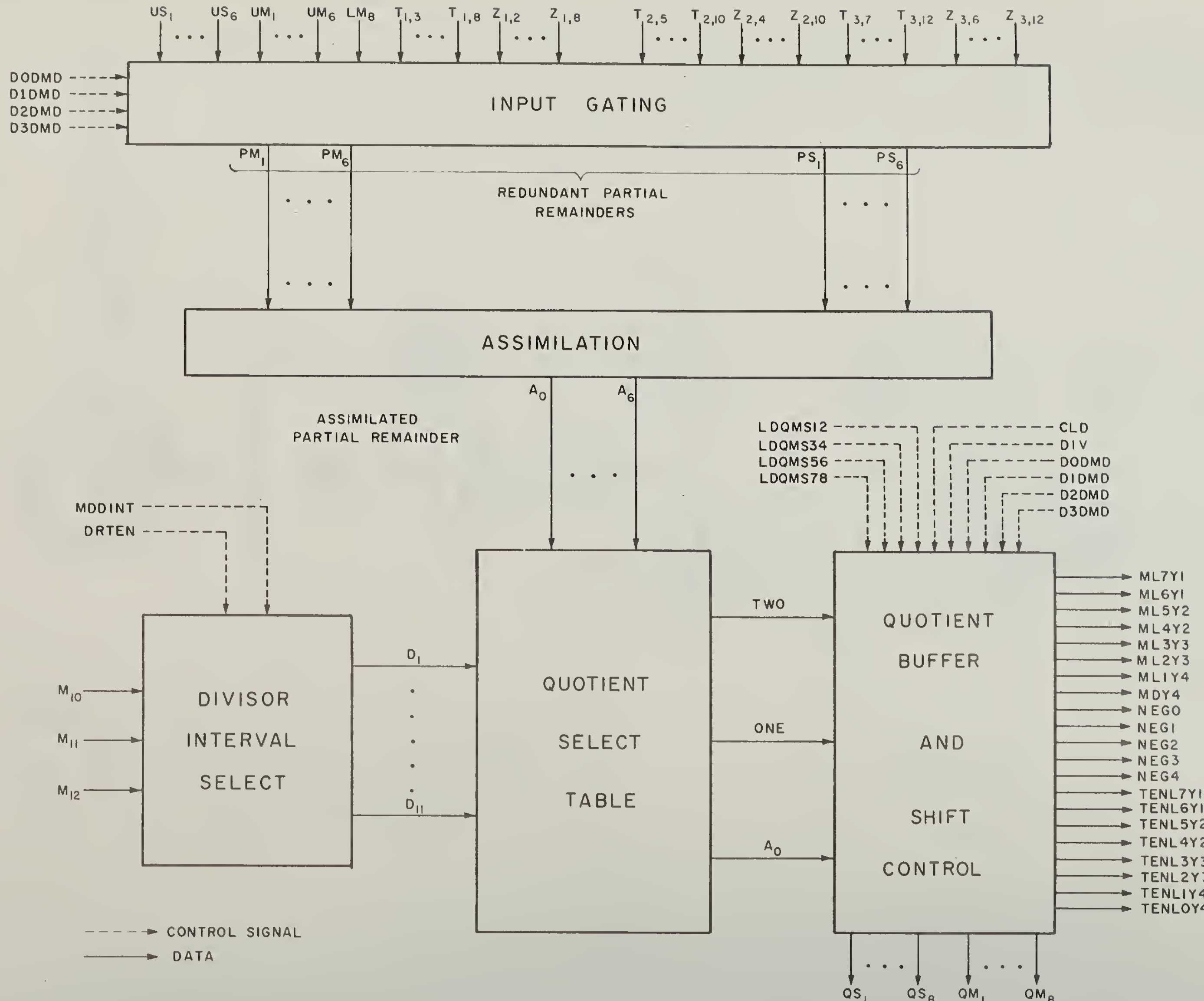
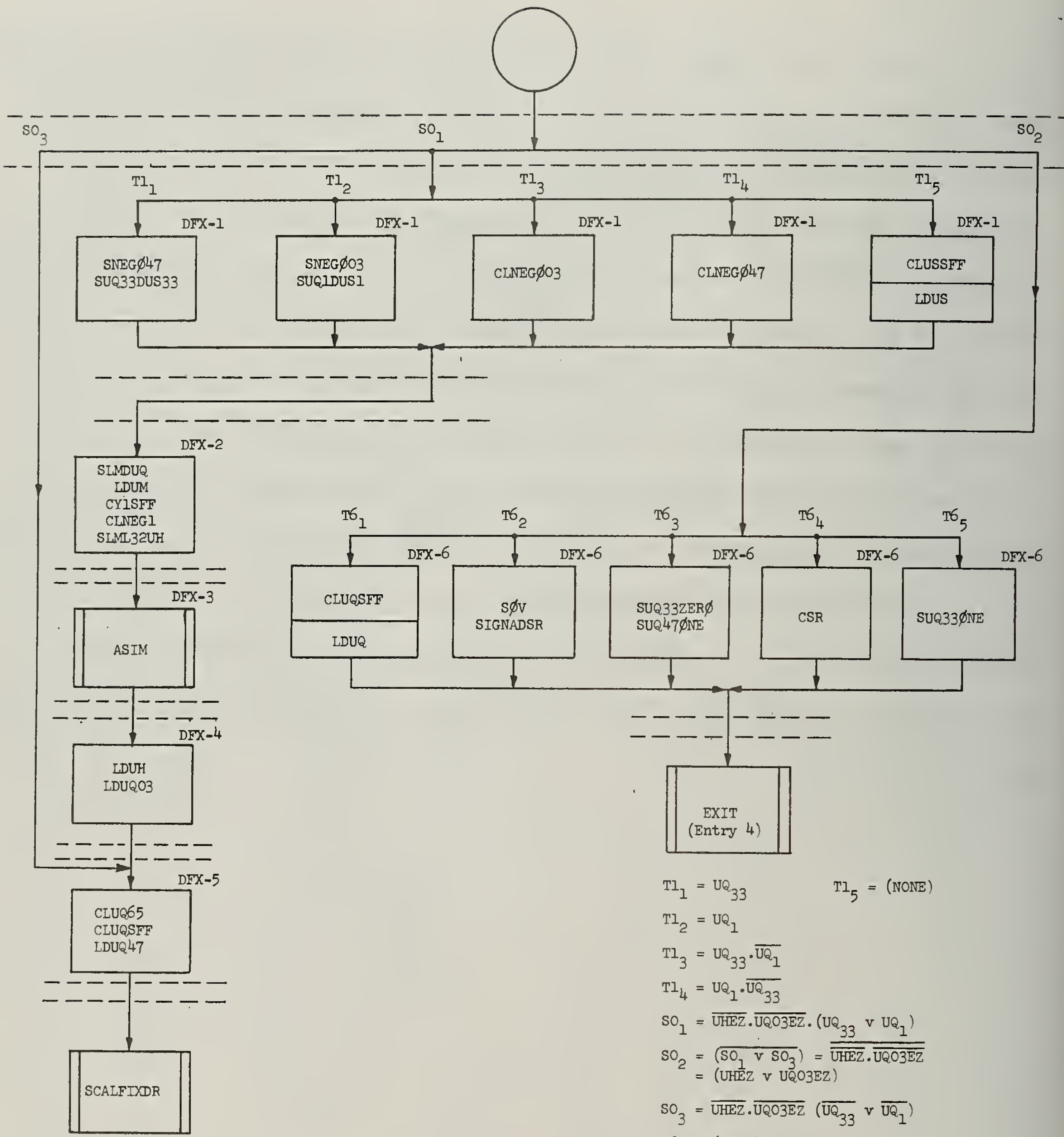


Figure 3.7.2.2.3. Block Diagram of MODEL DIVISION Logic

3.7.2.3 DFX

The control sequence DFX is used for fixed point division and comprises many procedure subsequences which are DIZETCOM, SCALFIXDR, FORMQAREM, COSREM, QASS and RESCALEREM. Although procedures by these very names do not exist in the simulation flow charts, but all the actions performed by these subsequences are present almost in the same time sequence as they occur in control point flow charts for DFX. There are hardly any significant differences, worth special mention, between these control point subsequences and flow charts and so the action of each task stage will not be specifically described here. A brief functional description of each of these control subsequences is already given in global flow description in Section 3.7.1. It is hoped that the reader would have no difficulty in understanding the control flow once he knows the functional interpretation of each control signal in the task stages of these control subsequences and also if he has carefully studied and understood the simulation flow charts.



$T1_1 = UQ_{33}$ $\qquad T1_5 = \text{(NONE)}$

$T1_2 = UQ_1$

$T1_3 = UQ_{33} \cdot \overline{UQ_1}$

$T1_4 = UQ_1 \cdot \overline{UQ_{33}}$

$SO_1 = \overline{UHEZ} . \overline{UQ03EZ} . (UQ_{33} \text{ v } UQ_1)$

$SO_2 = \overline{(SO_1 \text{ v } SO_3)} = \overline{\overline{UHEZ} . \overline{UQ03EZ}}$
$= (UHEZ \text{ v } UQ03EZ)$

$SO_3 = \overline{UHEZ} . \overline{UQ03EZ} \ (\overline{UQ_{33}} \text{ v } \overline{UQ_1})$

$T6_1 = \text{(NONE)}$

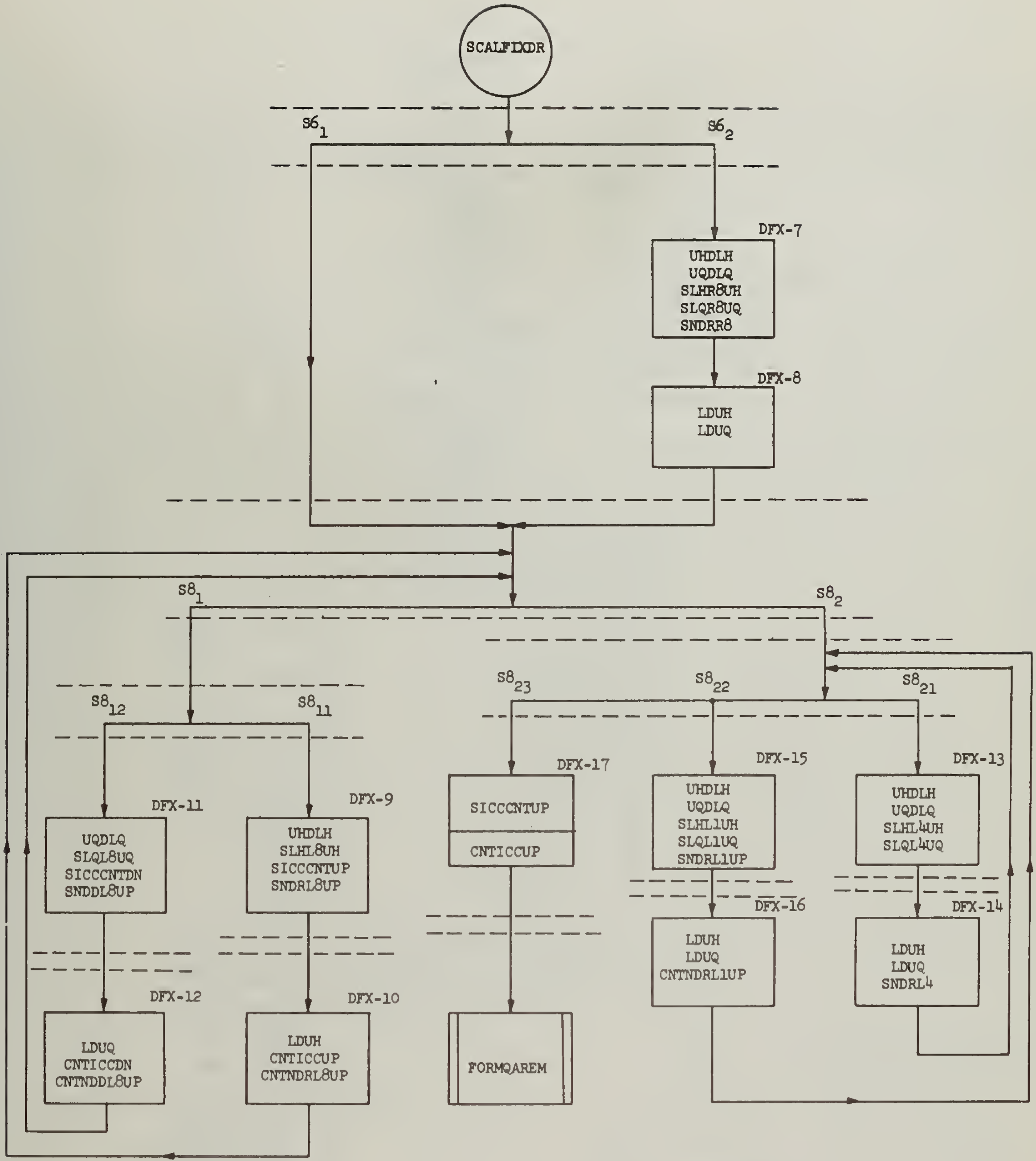
$T6_2 = UHEZ$

$T6_3 = UHEZ . \overline{SIGNA}$

$T6_4 = \overline{UHEZ}$

$T6_5 = UHEZ . SIGNA$

Figure 3.7.2.3.1. DFX_DIZETCOM—Fixed Point Division—
Divisor/Dividend Zero Test and 2's Complementation



$S6_1$ = UQ18EZ.UH18EZ

$S6_2$ = $\overline{S6_1}$

$S8_1$ = (UH916EZ v UQ916EZ)

$S8_2$ = $\overline{S8_1}$

$S8_{11}$ = UH916EZ

$S8_{12}$ = UQ916EZ.$\overline{\text{UH916EZ}}$

$S8_{21}$ = UH912EZ.$\overline{\text{UH1316EZ}}$

$S8_{22}$ = $\overline{\text{UH912EZ}}$.$\overline{UH_9}$

$S8_{23}$ = UH_9

UH916EZ = UH912EZ.UH1316EZ

UQ916EZ = UQ912EZ.UQ1316EZ

Figure 3.7.2.3.2. SCALFIXDR—Scale Fixed Point Divisor and Dividend

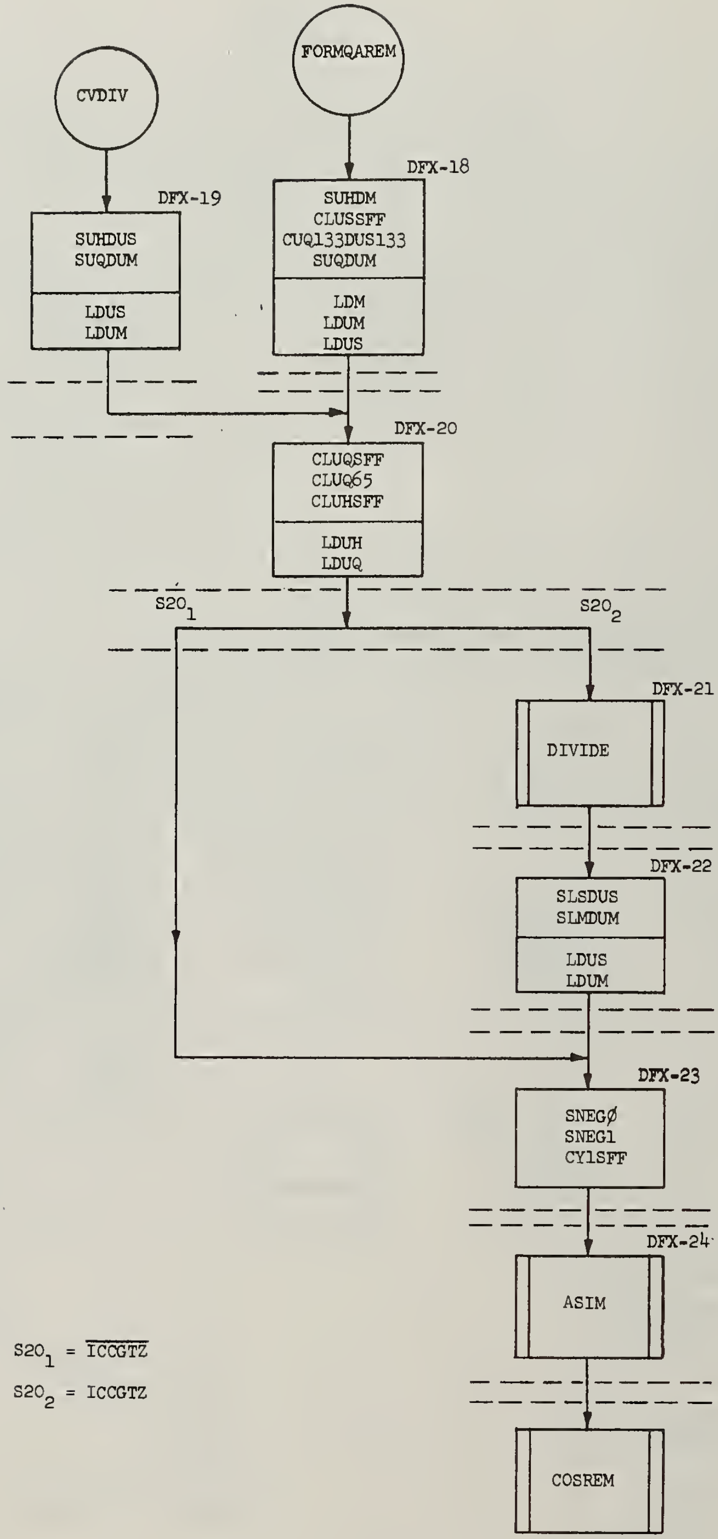
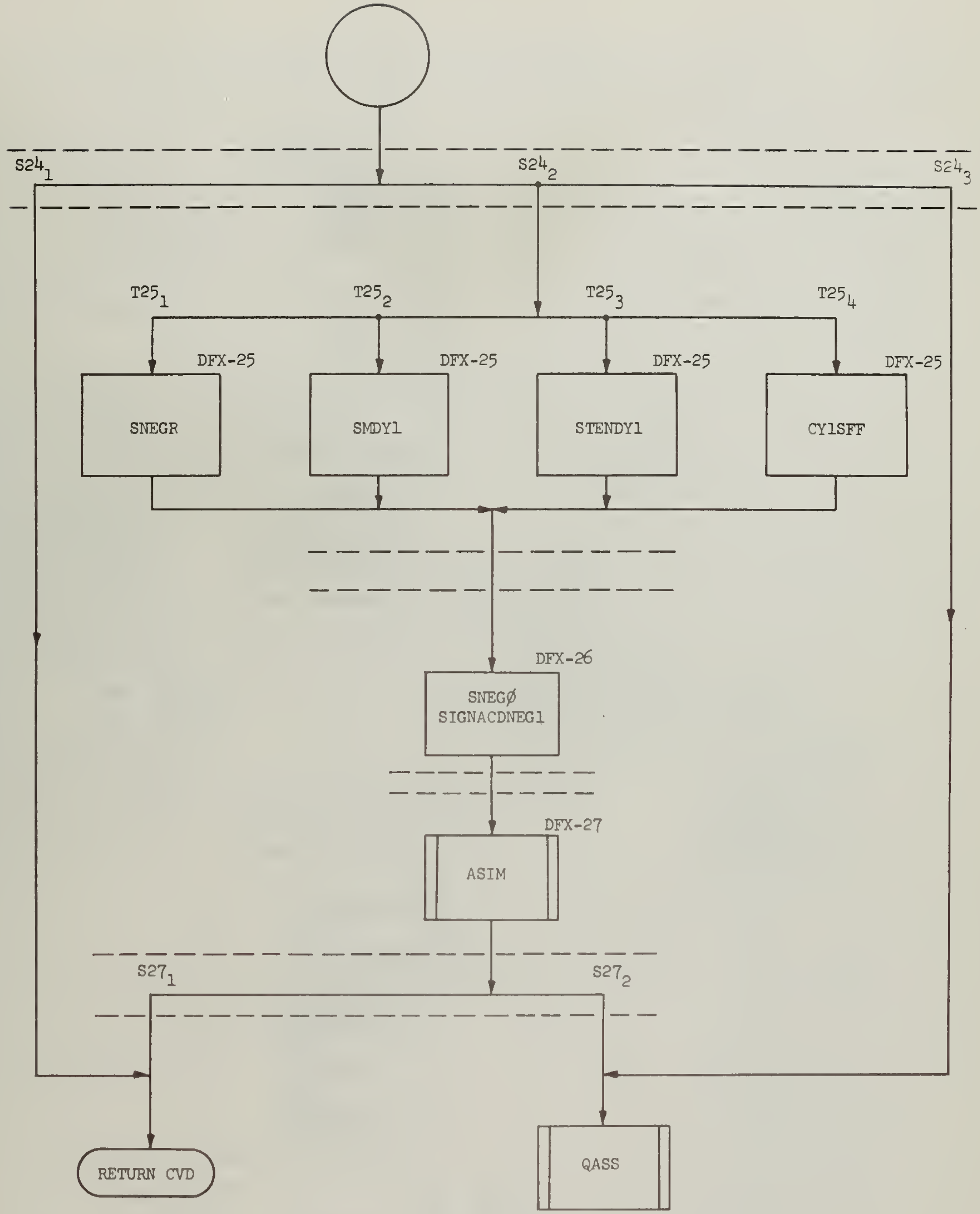


Figure 3.7.2.3.3. FORMQAREM—Form Quotient and Remainder



$S24_1 = \overline{LM_1}.\overline{SIGNA}.CVD$

$S24_2 = (LM_1 \vee SIGNA)$

$S24_3 = \overline{LM_1}.\overline{SIGNA}.\overline{CVD}$

$T25_1 = LM_1$

$T25_2 = LM_1.DIV$

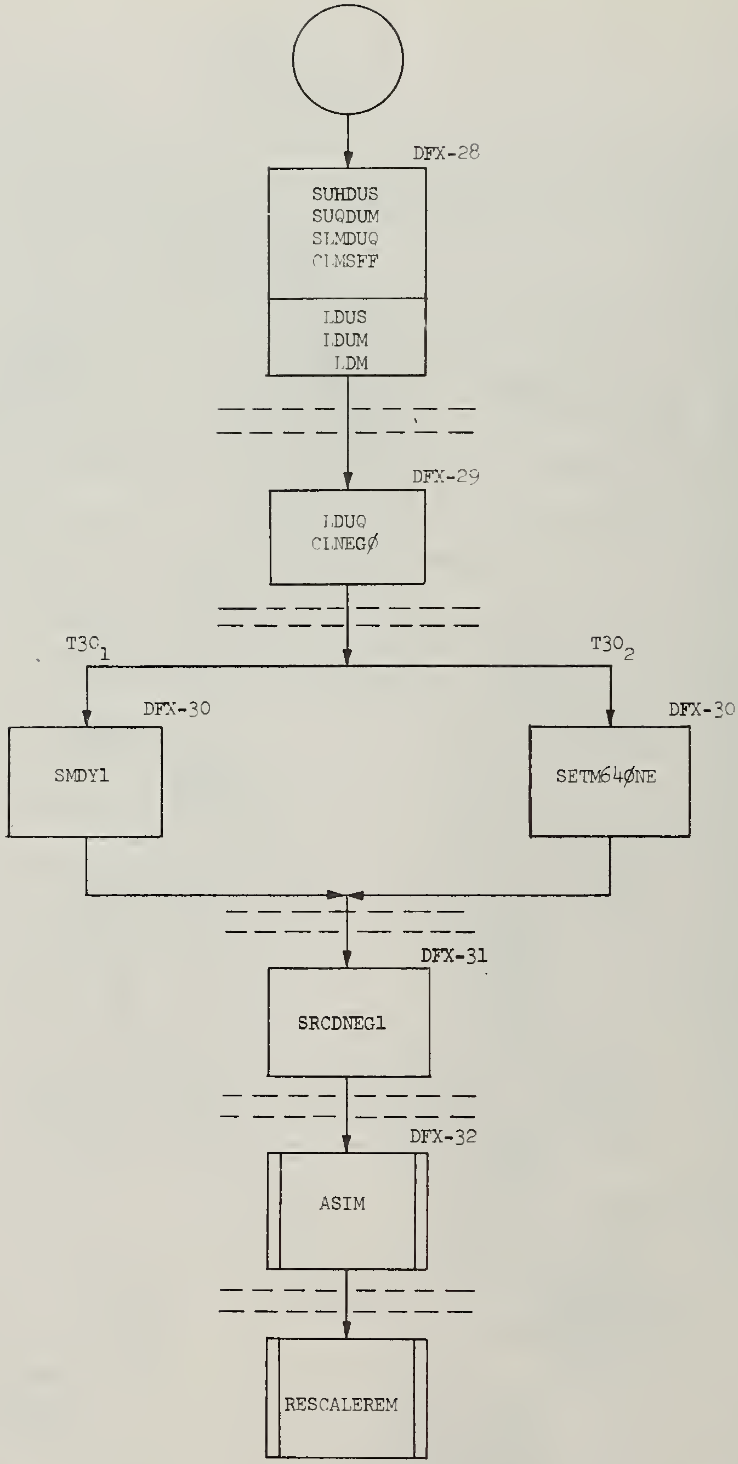
$T25_3 = LM_1.CVD$

$T25_4 = \overline{LM_1}.SIGNA$

$S27_1 = CVD$

$S27_2 = \overline{S27_1}$

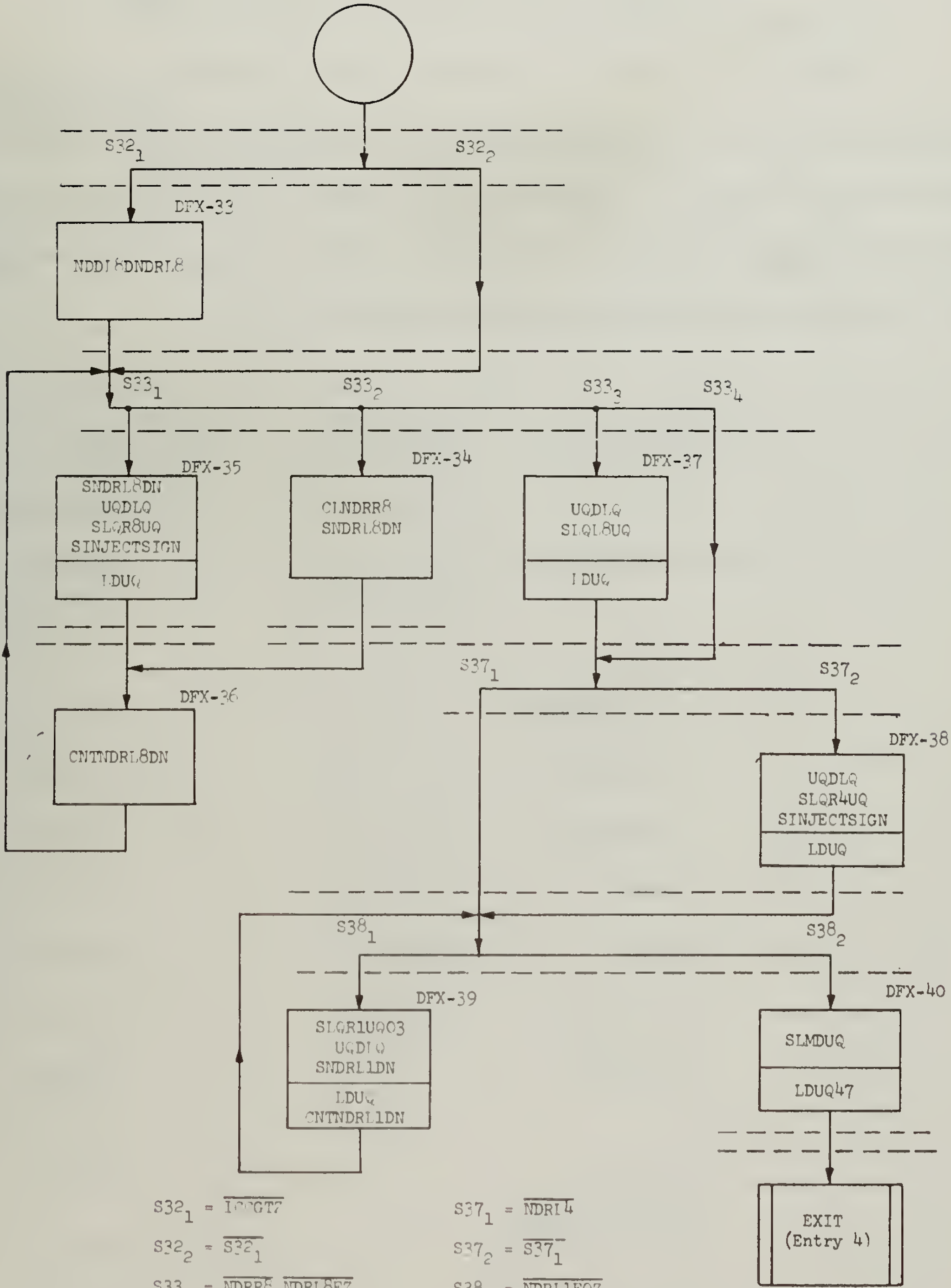
Figure 3.7.2.3.4. COSREM—Correct Sign of Remainder



$T30_1 = \overline{NEGR}$

$T30_2 = NEGR$

Figure 3.7.2.3.5. QASS—Quotient Assimilation into Conventional Representation



$S32_1 = \overline{\text{I??GT7}}$

$S32_2 = \overline{S32_1}$

$S33_1 = \overline{\text{NDRR8}}.\overline{\text{NDRL8EZ}}$

$S33_2 = \text{NDRR8}.\overline{\text{NDRL8EZ}}$

$S33_3 = \text{NDRR8}.\text{NDRL8EZ}$

$S33_4 = \overline{\text{NDRR8}}.\text{NDRL8EZ}$

$S37_1 = \overline{\text{NDRL4}}$

$S37_2 = \overline{S37_1}$

$S38_1 = \overline{\text{NDRL1EQZ}}$

$S38_2 = \overline{S38_1}$

Figure 3.7.2.3.6. RESCALEREM—Remainder Postscaling

3.7.3 Logic Implementation

Table below shows the various control subsequences necessary for the DIV control sequence for both fixed point and floating point operands. It also shows the figure number of their control point flow charts and the number of corresponding logic drawing on which these flow charts are implemented. Besides, this also shows the drawing numbers of TSC drawings.

Control Subsequence Name	CP flowchart Figure No.	Corres. Logic Drawing No.
DFL	3.7.2.1	AUO-07-371-01
DIVIDE	3.7.2.2.1 3.7.2.2.2	-371-02 -371-02, 03
ASIM	3.5.2.3	-371-04
DFX-DIZETCOM	3.7.2.3.1	-371-04, 05
SCALFIXDR	3.7.2.3.2	-371-08
FORMQAREM	3.7.2.3.3	-371-08
COSREM	3.7.2.3.4	-371-09A
QASS	3.7.2.3.5	-371-09B
RESCALEREM	3.7.2.3.6	-371-10, 11
REFOTRAN		-332-05 -352-04, 05, 06
Counters		-371-15
Task Signal Collector "B_1"		-371-12
Task Signal Collector "B_2"		-371-13
Task Signal Collector "B_3"		-371-14

3.8 NUMBER SYSTEM CONVERSION CONTROL SEQUENCES

3.8.1 Introduction

This section describes control sequences which are used to transform an operand of a given number type into an equivalent operand of another specified number type. These sequences are CVL (Convert to Long Fixed Point), CVF (Convert to floating point) and CVD (Convert to Decimal). Each of these control sequence corresponds to one arithmetic order of the same name. Each control sequence is broken up into many Procedure Subsequences which are functionally meaningful and ease the comprehension of the control flow. In many cases, these subsequences bear the same name as the various procedures in simulation flow charts.

3.8.2 Conversion to Long Fixed Point (CVL) Control Sequence

This control sequence is used to convert either a floating point number or a decimal number into a long-fixed point number. If the converted fixed number is greater than $(2^{31} -1)$ or less than (-2^{-31}), an overflow will occur and the result returned to TP will be the low order 31 bits with correct sign bit.

3.8.2.1 Global Flow Description

Figure 3.8.2.1 shows the global flow diagram for the CVL control sequence. As can be seen from the flow diagram, there are two distinct types of CVL: floating point to long-fixed point number and decimal to long-fixed point number.

In order to convert a floating point operand into long-fixed type, the subsequence CVL-FLT in entered by the control. Since the exponent is represented in excess 64, this sequence subtracts 64 from the exponent and then checks whether the long-fixed-point number equivalent to the given floating point number can be represented in one word. If the absolute value of the given floating point number is less than 1 or greater than 16^{21}, the converted number is zero and accordingly the result register is set to zero. If the absolute value $|x|$ is $16^{8} < |x| < 16^{21}$ i.e. if the exponent lies between 8 and 21, converted number will overflow but will possess significant digits so that low-order 31 bits with correct sign are transmitted to the process along with an overflow indicator flag. In case the exponent is less than or equal to 21, the CVL-FLT control subsequence calls the subsequence FL-NØRM-FX which converts the floating point number to a double word fixed point number by shifting right or left the contents of UQ register according to the exponent of the floating point number. The control sequence FL-NØRM-FX has exactly the same functional significance as the same procedure in simulation flow chart.

If the floating point number is negative, then the converted long-fixed point number is converted to 2's complement form by entering the control subsequence NEGATE which calls upon the subroutine control subsequence COMPL.

Then the control passes on to subsequence LUQ which left adjusts the UQ. Finally, the control branches to terminal sequence REFOTRAN.

However, if the given number to be converted is decimal, the control enters the subsequence CVL-DEC which calls upon the subroutine control sequence DVB to convert the decimal digits into binary bits which are right adjusted in register UQ. The control sequence DVB itself makes use of subroutine control sequence ASIM. Again, if the given number is negative, the converted number obtained from subsequences DVB is transformed to 2's complement representation. by the subsequence NEGATE. Then the control passes on to subsequences LUQ which left adjusts the contents of UQ so that the first word transmitted out of UQ to TP will contain the necessary converted number and the flag indicators.

Finally, the control branches to terminal sequence REFOTRAN via subsequence EXIT, which transmits the result to the processor. The processor will ignore the second word of UQ transmitted in subsequence X-ØUT because this word contains no useful information.

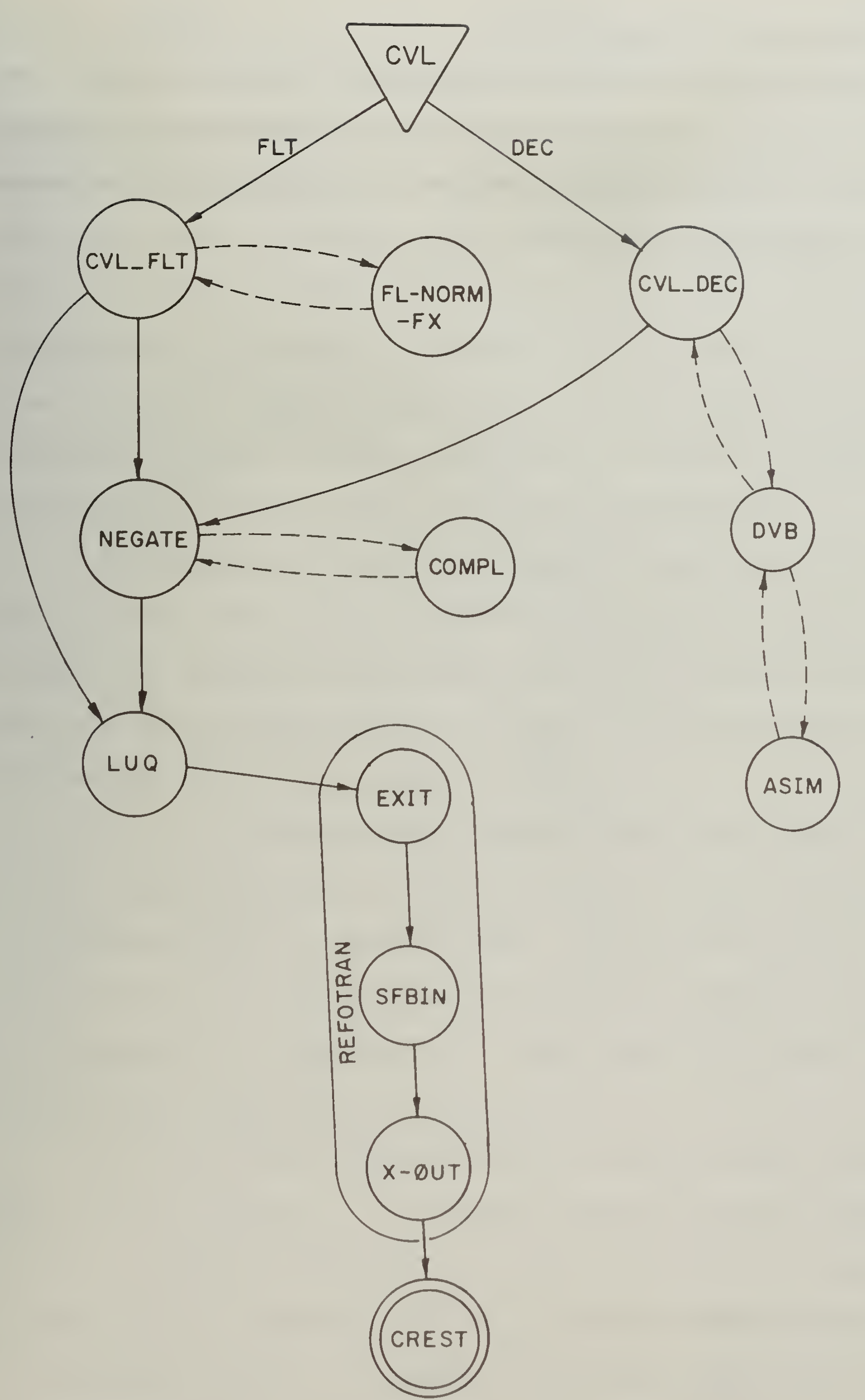


Figure 3.8.2.1. Global Flow Diagram for "CVL--Conversion to Long Fixed Point" Control Sequence

3.8.2.2 Control-Point Flow Description

When the given operand to be converted is floating point type, the number is in register UQ, bits 9 through 64 and the exponent is in register EUU, in contrast to the simulation flow chart and the sign bit of the given operand is in SIGNA control flip-flop.

3.8.2.2.1 CVL-FLT

The control enters the subsequence CVL-FLT whose control point flow chart is shown in Figure 3.8.2.2.1.1. Task stages CVL-1 and CVL-2 transfer the contents of EUU i.e. the exponent of the operand to the counter register EUL through the exponent adder path. This transfer is necessary because the exponent 'magnitude check logic' is at the output of counter-register EUL only. Note that no subtraction of 64 is done in control hardware because the exponent magnitude check logic can handle excess 64 representation easily. Secondly, the sum or differenee of exponents in registers EUU and EUM available in EUL is always in excess 64 representation.

Sequence stage S2 checks the exponent to see whether the converted number will be all zero ($S2_1$ i.e. whether the exponent is $\leqslant 0$ and >21), whether the number will overflow but have significant digits (condition $S2_1$) or whether the converted number will be in range for full significance representation (condition $S2_3$ i.e. the $1 \leqslant$ exponent $\leqslant 8$).

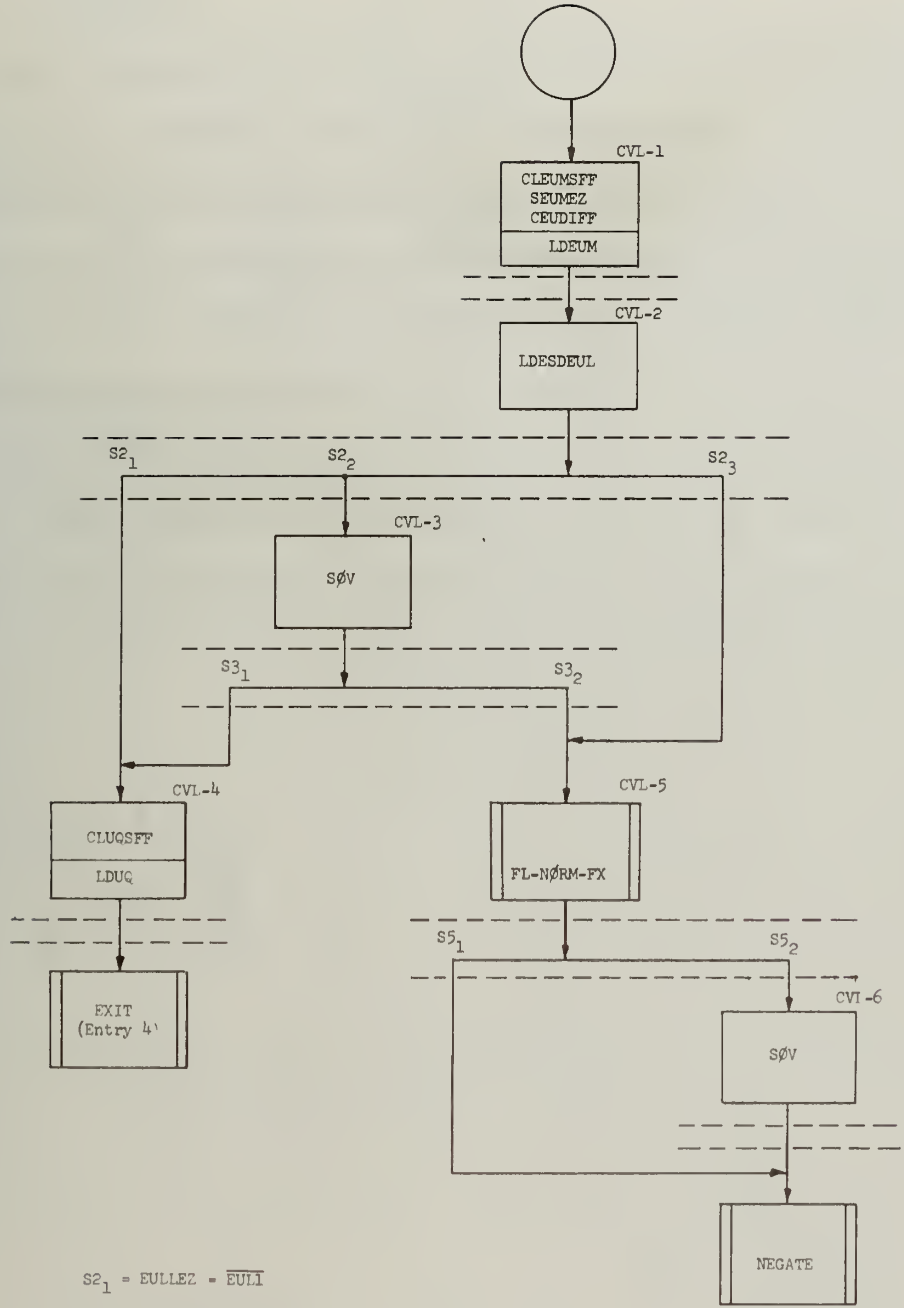
In case of converted number being all zero, task stage CVL-4 clears the result register UQ and the control then goes to terminal sequence REFOTRAN via subsequences EXIT. For overflow cases, task stage CVL-3 sets the control flip-flop ϕV and then branches to task stage CVL-4 if exponent is > 21 or to the task stage CVL-5 if exponent is $\leqslant 21$. Task stage CVL-5 is also entered from the sequence stage S2 under condition $S2_3$.

Task stage CVL-5 calls the subroutine control sequence FL-NØRM-FX shown in Figure 3.8.2.2.2.1. This sequence shifts the number in UQ either to the right if exponent is less than 14, to the left if exponent is greater than 14 or none at all if exponent is 14. At the end of FL-NØRM-FX control sequence, the floating point number is changed to an equivalent double word fixed-point number in register UQ.

Sequence stage S5 checks whether the converted number has overflowed (i.e. number is $>(2^{31}-1)$ or $<(-2^{31})$. Task stage CVL-6 sets the control flip-flop ϕV in case of overflow, and then the control passes to procedure subsequence NEGATE shown in Figure 3.8.2.2.1.2. If the given number to be converted is positive, the NEGATE subsequence is bypassed and the control goes to subsequence LUQ, which left adjusts the contents of UQ. If however, the number is negative, Task stage CVL-10 clears the register US and sets the negate control signal $\overline{\text{NEGØ}}$ so as to complement the output of UM.

Task stage CVL-11 calls the subroutine control sequence COMPL where the contents of UQ are complemented with the help of SDS-array, assimilated into conventional form and then transferred to the result register UQ. Control point flow chart for COMPL is shown in Figure 3.8.2.2.5.1. At the end of NEGATE subsequence, the converted number is available in result register UQ but right justified. However, since the TP expects the converted number in the very first word transmitted to it, and secondly since the flag indicators ϕV and ID are transmitted always with the first word, the contents of UQ must be left adjusted. So the control passes to the procedure control subsequence LUQ whose control point flow chart is shown in Figure 3.8.2.2.4.1 and is explained in Section 3.8.2.2.4.

After the converted number is left adjusted in result output register UQ, the control finally passes to the terminal sequence REFOTRAN via the subsequence EXIT.



$S2_1 = EULLEZ = \overline{EUL1}$

$S2_2 = EULGT8 \vee EULGT21$

$S2_3 = \overline{S2_1 \vee S2_2}$

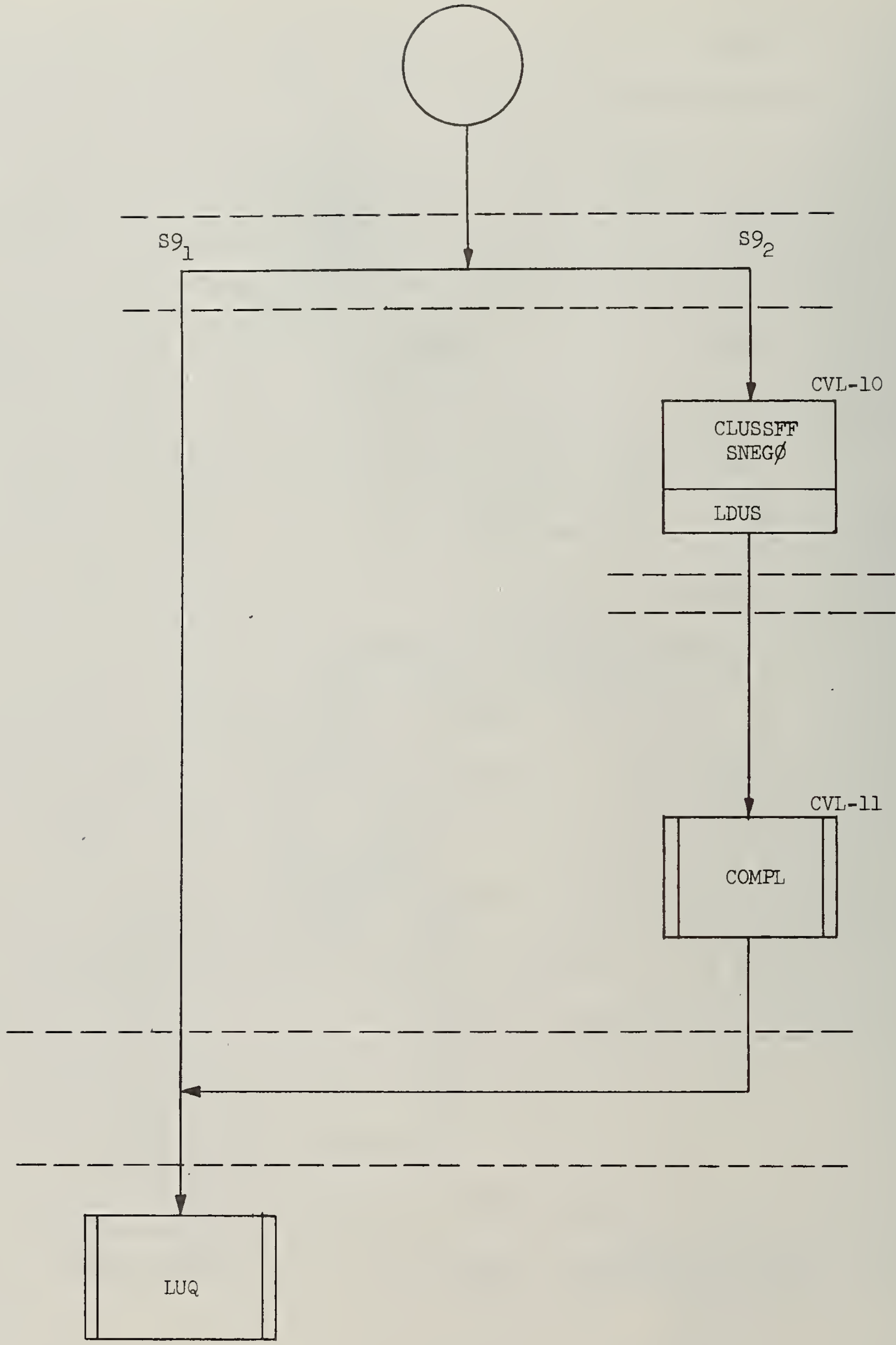
$S3_1 = EULGT21$

$S3_2 = EULGT8.\overline{EULGT21}$

$S5_1 = \overline{UQ_{33}} \vee SIGNA = \overline{S5_2}$

$S5_2 = UQ_{33}.\overline{SIGNA}$

Figure 3.8.2.2.1.1. CVL_FLT—CVL Floating Point Operand



$S9_1 = \overline{S9_2}$

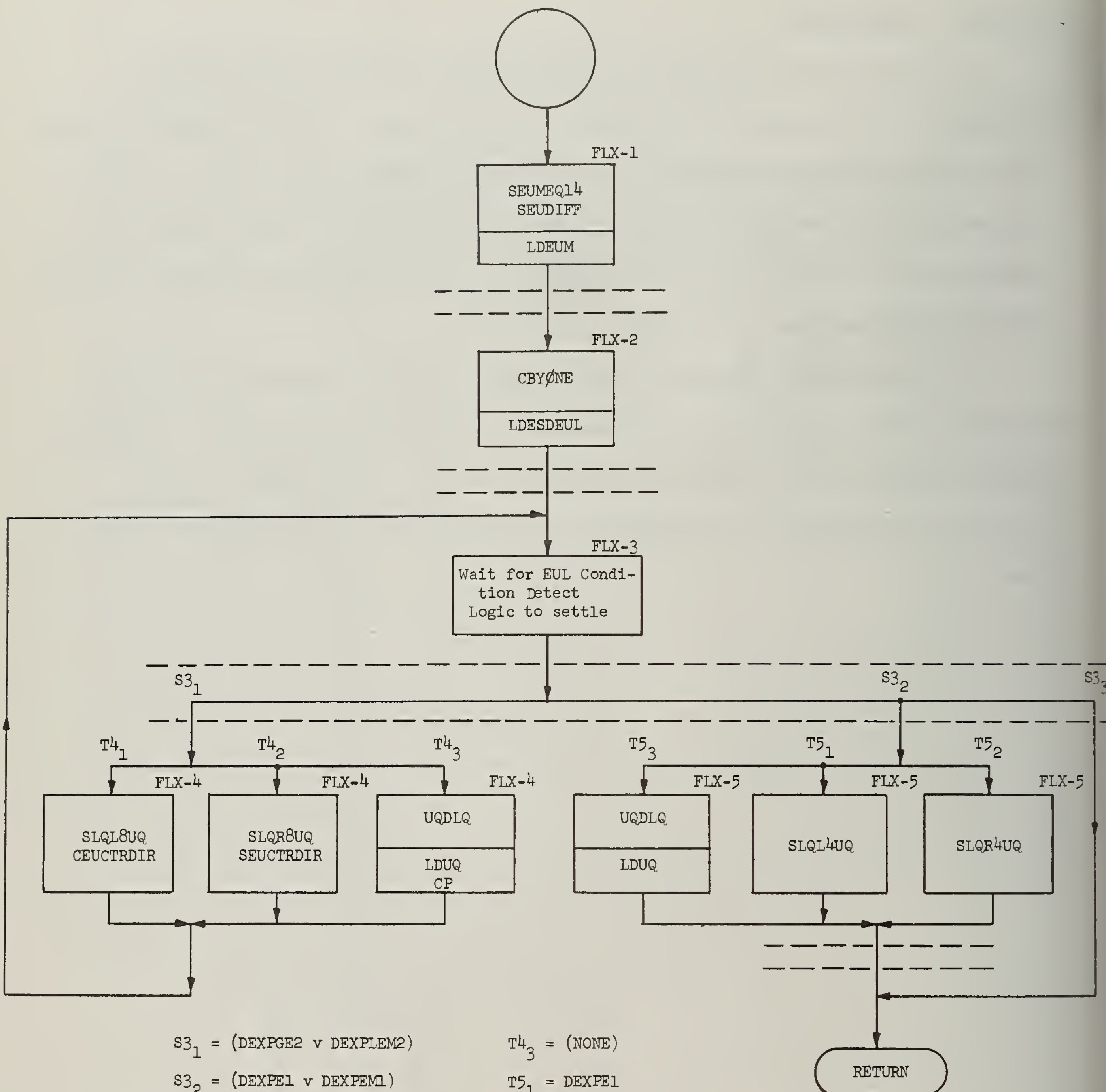
$S9_2 = SIGNA$

Figure 3.8.2.2.1.2. NEGATE--Negate a Positive Number

3.8.2.2.2 FL-NØRM-FX

This sequence is shown in Figure 3.8.2.2.2.1 and is absolutely identical to the simulation flow chart of the same name. Its functional description is already fiven in global flow description. Task stages FLX-1 and FLX-2 decrease the exponent in EUL by 14. Task stage FLX-3 is a dummy stage with no task signals but acts only as a time buffer for the sequence condition, obtained from the 'EUL condition detect logic', to be stable.

Task stage FLX-4 shifts the contents of UQ by eight bits either to the left or right according as the exponent difference is positive or negative, where Task-stage FLX-5 shifts the same by four bits. The number of shifts necessary is dependent on the difference as obtained in Task stages FLX-1 and FLX-2.



$S3_1$ = (DEXPGE2 v DEXPLEM2)

$S3_2$ = (DEXPE1 v DEXPEM1)

$S3_3$ = $\overline{(S3_1 \text{ v } S3_2)}$

$T4_1$ = DEXPGE2

$T4_2$ = DEXPLEM2

$T4_3$ = (NONE)

$T5_1$ = DEXPE1

$T5_2$ = DEXPEM1

$T5_3$ = (NONE)

Figure 3.8.2.2.2.1. FL_NØRM_FX—Convert a Floating Point Operand to Fixed Point Operand

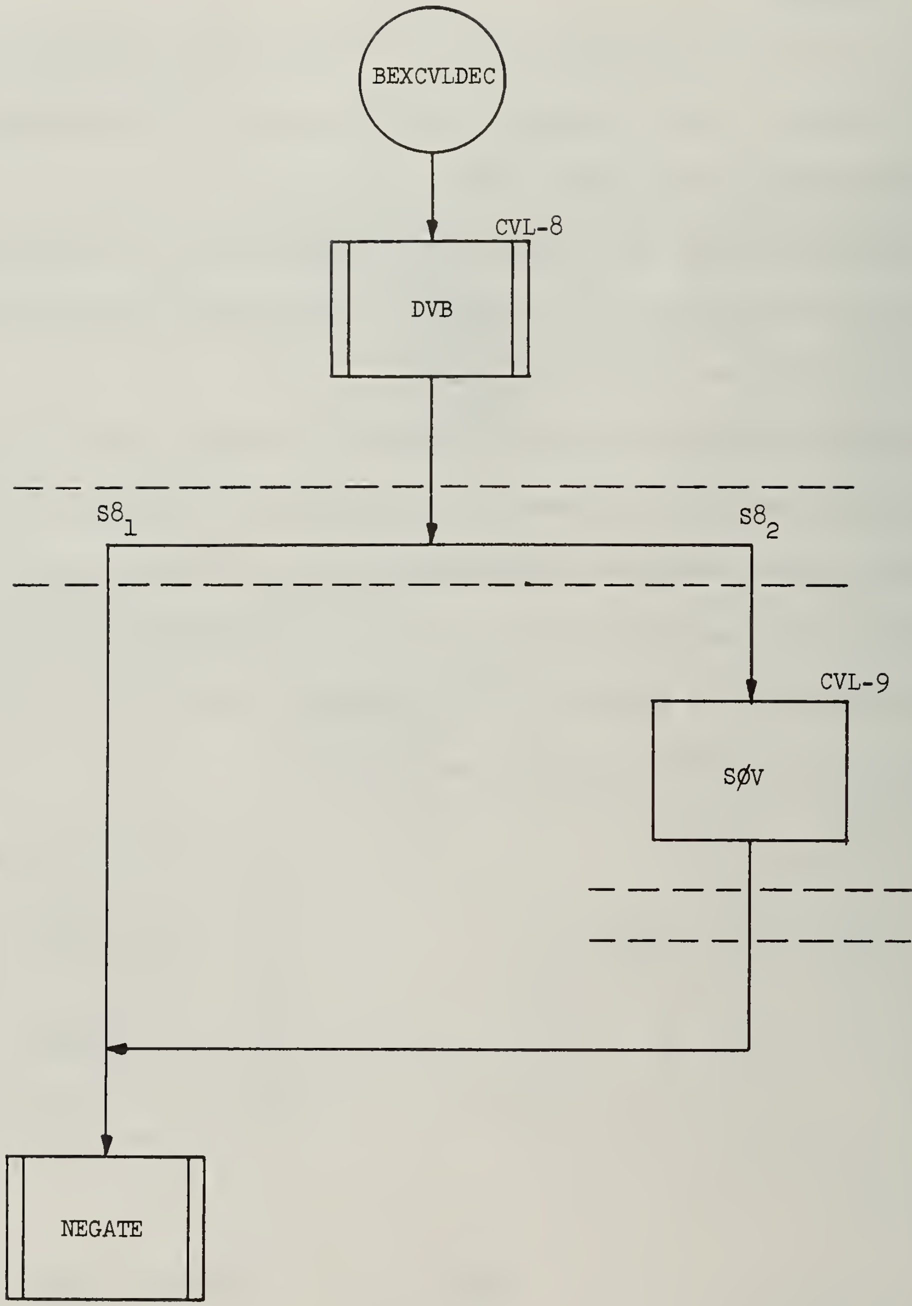
3.8.2.2.3 <u>CVL-DEC</u>

The control point flow chart for this control subsequence is shown in Figure 3.8.2.2.3.1 and is entered when the number to be converted to long-fixed point format is of decimal type.

The task stage CVL-8 calls the subroutine control sequence DVB which converts the decimal digits into binary bits which are right justified in UQ. For details of DVB, see Section 3.8.2.2.3.1.

Then the sequence stage S8 checks for overflow condition in the converted number and either bypasses the Task stage CVL-9 which sets control flip-flop ϕV or goes through it depending upon the overflow condition.

Now if the original number to be converted was negative, then this converted number must be changed to 2's complement form as was done in case of floating point operands explained earlier.



$S8_1$ = UQ03EZ

$S8_2 = \overline{S8_1} = \overline{\text{UQ03EZ}}$

Figure 3.8.2.2.3.1. CVL_DEC--CVL Decimal Operand

3.8.2.2.3.1 DVB

Figures 3.8.2.2.3.2 and 3.8.2.2.3.3 show the control point flow charts for the subroutine control subsequence DVB. It is used to convert the decimal digits, stored in bits 9-64 of UQ register, into binary bits, into binary bits which are right adjusted in UQ. This routine is used both both by CVL and CVF whenever the number type is decimal.

The control point flow chart is exactly identical to the simulation flow chart of the same name. The only difference is the first task stage DVB-1 which is really a dummy control point with no task signal. However, it does perform a useful hardware function in generating the return signal if the number to be converted is all zero. This control point simplifies the generation of reply signal from the DVB subroutine control subsequence.

Task stage DVB-2 sets the exponent counter register EUL to 14 and in a downward counting direction with a step down of 2 in each counting.

Task stage DVB-3 shifts the contents of UQ to get rid of leading zero decimal digits. After the leading zeros are gone, DVB-4 transfers the first nonzero decimal digit to be converted to register M, and the control goes to the conversion loop's first Task stage DVB-5.

DVB-5 clears registers US, UM and transfers second most significant decimal digit to register UM. Task stage DVB-6 sets up the SDS-array and M-array such that the digit in register M is multiplied by 10 and added to the digit in UM. The result is available at the output of SDS4 from where it is transferred back again to register UM, US to do the assimilation of the sum obtained. Loop counter is decremented by unity and SDS-array is prepared for subroutine control sequence ASIM.

Task stage DVB-8 calls the control subsequence ASIM which assimilates the conversion result of preceeding digits into conventional form. Sequence stage S8 checks if all the digits have been converted. If not, contents of UQ are lect shifted by 4 bits, next decimal digit is transferred to M and the control loops back to Task stage DVB-5 to begin another conversion pass. If, however, all the digits have been converted, the assimilated and converted output is available in LM from where it is transferred right justified to result register UQ by Task stage DVB-10.

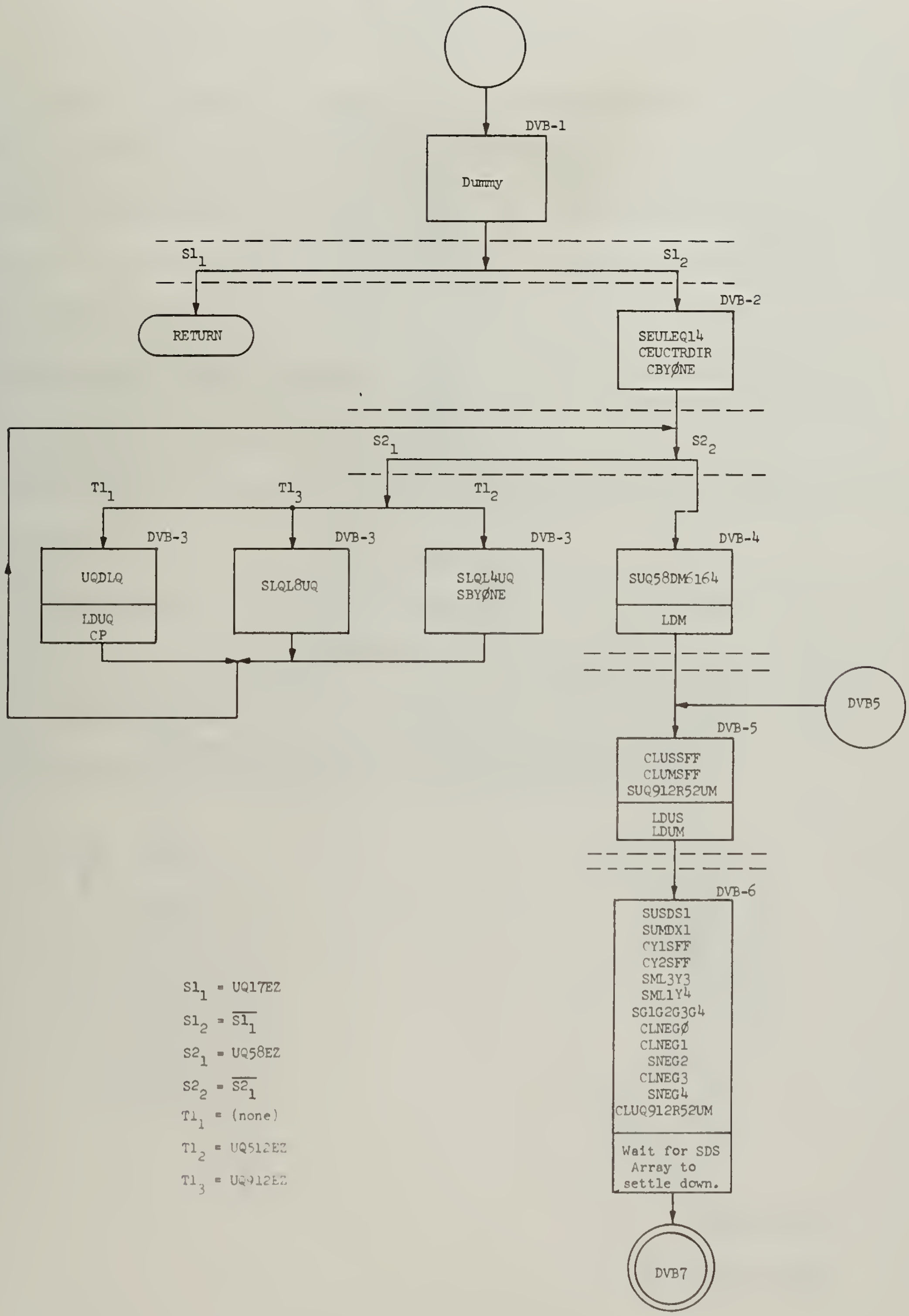
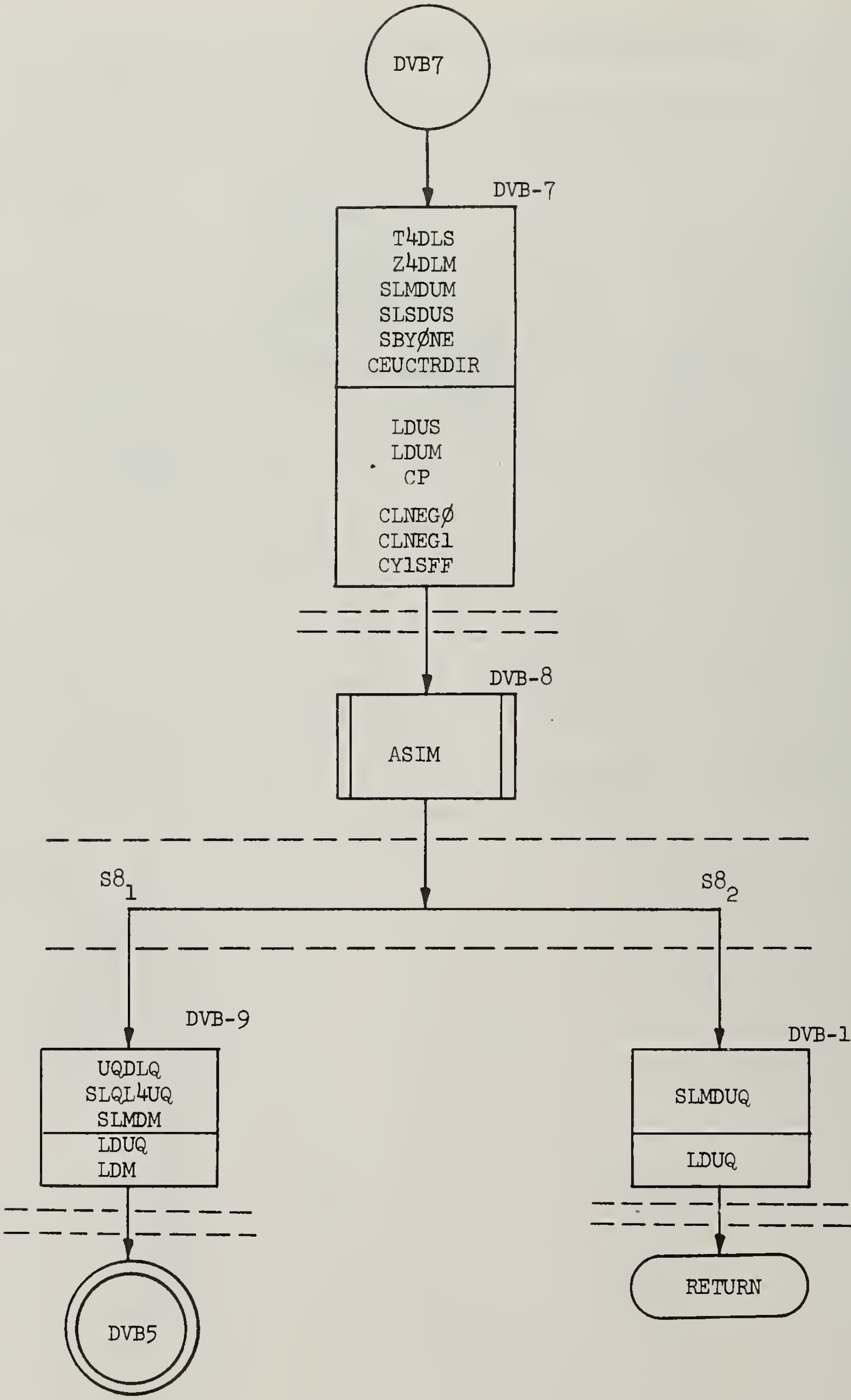


Figure 3.8.2.2.3.2. DVB--Decimal to Binary Conversion



$S8_1 = \overline{EULEZ}$

$S8_2 = EULEZ = \overline{S8_1}$

Figure 3.8.2.2.3.3. DVB--Decimal to Binary Conversion

3.8.2.2.4 LUQ

Control point flow chart for this procedure control subsequence is shown in Figure 3.8.2.2.4.1. This subsequence left adjusts the one word of data lying in the last four bytes of 8 byte long result register UQ.

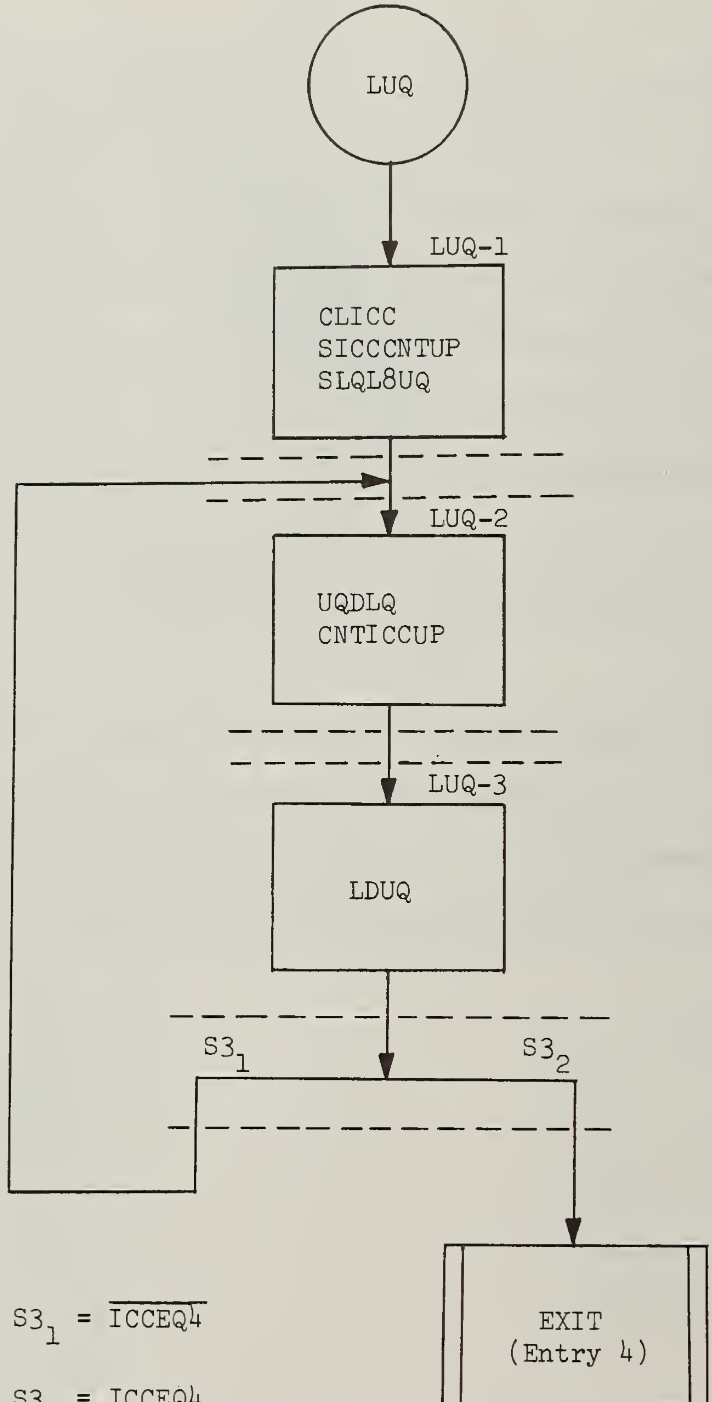
Task stage LUQ-1 clears the byte shift bidirectional counter ICC and also sets it to the upward counting state. Besides, it sets up the left shift path between the register LQ and register UQ.

Task stage LUQ-2 transfers contents of UQ without any shift to register LQ and updates the byte shift counter by unity.

Task stage LUQ-3 gates the contents of LQ shifted 8 bits to the left into register UQ.

Sequence stage S3 checks if the total shift of 4 bytes has taken place. If not, the control loopsback to the input of Task stage LUQ-2 and the process is repeated till sequence stage S3 detects the end of the loop

Finally the control passes to the subsequence EXIT.



$S3_1 = \overline{ICCEQ4}$

$S3_2 = ICCEQ4$

$= \overline{S3_1}$

Figure 3.8.2.2.4.1 LUQ—Left Adjust Contents of Register UQ

3.8.2.2.5 COMPL

Figure 3.8.2.2.5.1 shows the control point flow chart of control subsequence COMPL. This is used to do 2's complementation of the contents of UQ. This is very similar to the simulation flow chart and is fairly simple and so needs no special explanation.

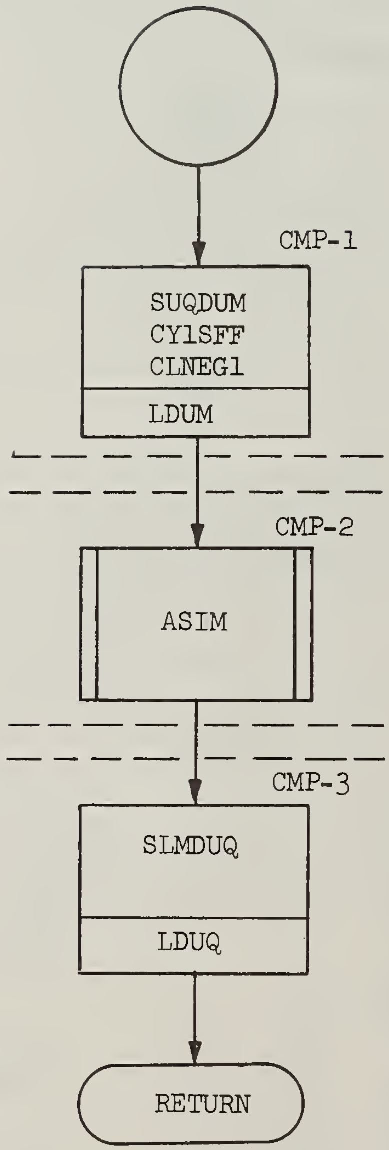


Figure 3.8.2.2.5.1. COMPL—Form 2's Complement of the Contents of Register UQ

3.8.2.3 Logic Implementation

Table below lists the names of various control subsequences necessary for CVL control sequence, the figure numbers of their control point flow charts and the number of the corresponding logic drawing which implements the control subsequence.

Control Subsequence Name	CP flow chart Figure No.	Corres. Logic Drawing No.
CVL-FLT	3.8.2.2.1.1	AUO-07-382-01
FL-NØRM-FX	3.8.2.2.2.1	382-03
CVL-DEC	3.8.2.2.3.1	382-04
DVB	3.8.2.2.3.2 3.8.2.2.3.3	382-04, 05
ASIM	3.5.2.3	371-04
NEGATE	3.8.2.2.1.2	382-02
COMPL	3.8.2.2.5.1	382-05
LUQ	3.8.2.2.4.1	382-02A
REFOTRAN		332-05 352-04, 05, 06
Task Signal Collector "C_1"		382-06
Task Signal Collector "C_2"		382-07

3.8.3. CONVERSION TO FLOATING POINT (CVF) CONTROL SEQUENCE

This control sequence is used to convert either a fixed point number (both long and short) or a decimal number into a floating point number. It shares a few control subsequences which are also used by other conversion instructions.

3.8.3.1 Global Flow Description

Figure 3.8.3.1.1 shows the global flow diagram for CVF control sequence. It mainly consists of one subsequence CVF-DEFIX which calls upon other subroutine control subsequences DVB, COMPL and NØRMUQ, besides the terminal sequence REFOTRAN.

The control subsequence CVF-DEFIX calls upon the subroutine sequence DVB to convert the given decimal number to binary. It then loads the proper exponent in the exponent register. Later,it normalizes the mantissa of the converted number before entering the terminal sequence REFOTRAN.

If, however, the number to be converted is in fixed point format, this control sequence provides control signals to load the proper exponent in the exponent register EUL, to shift the register UQ in order to share the subroutine control sequence NØRMUQ. It also complements the number, in case it is negative, so that the mantissa of the resulting floating point number be positive. After complementation and normalization, the control enters finally the terminal sequence REFOTRAN.

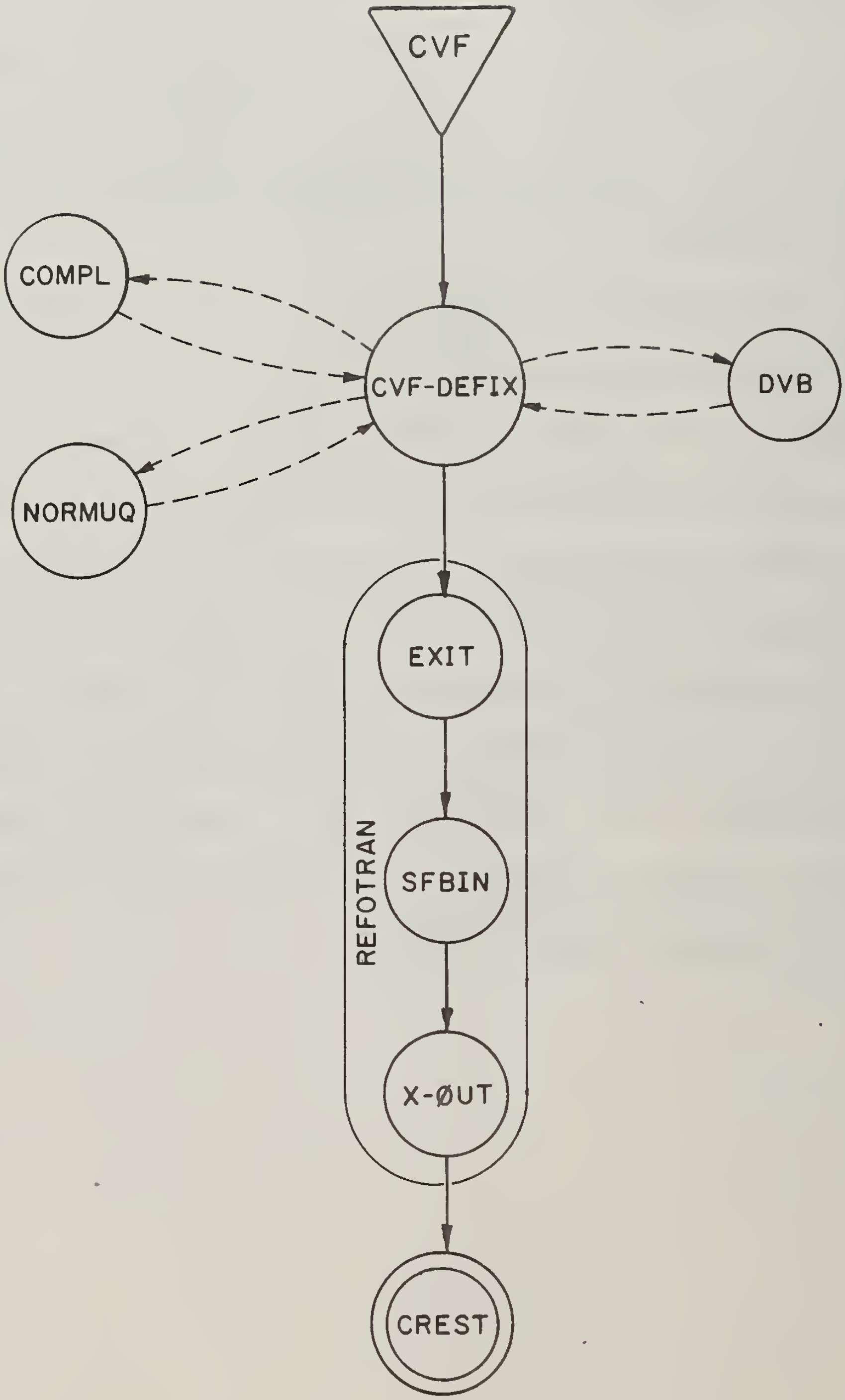


Figure 3.8.3.1.1. Global Flow Diagram for "CVF--Conversion to Floating Point" Control Sequence

3.8.3.2 Control-Point Flow Description

Figure 3.8.3.2.1 shows the control point flow chart for the subsequence CVF-DEFIX. This subsequence has two entries: entry CVFIX for fixed point operands and entry CVFDEC for decimal operands.

For decimal operands, the number to be converted is in UQ register bits 9 through 64 and the sign bit is in control flip-flop SIGNA. Task stage CFD-1 calls the subroutine control sequence DVB described earlier in Section 3.8.2.2.3.1 which converts the decimal number in register UQ into an equivalent binary number which is right-adjusted in result register UQ. Then the task stage CFD-2 loads the exponent result register EUL with binary number equivalent to decimal 14, in excess 64 representation.

Task stage CFD-3 calls upon the subroutine control sequence NØRMUQ which normalizes the converted number to lie between 1/16 and 1. Control point flow chart for subsequence NØRMUQ is shown in Figure 3.4.2.1.1.

Task stage CFD-4 loads the control flip-flop SR (which gives the sign of the result) with the sign of the number to be converted and finally the control goes to terminal sequence REFOTRAN via subsequence EXIT.

For fixed-point operands, the number to be converted is in register UQ bits 1 through 32. Task stage CFX-1 loads the exponent result register EUL with an excess 64 binary representation of decimal 8 and at the same time shifts the contents of UQ right by 8 bits so that it can share the subroutine control sequences COMPL and NØRMUQ. If the given fixed-point number is negative (sequence condition $S1_1$), the task stages CFX-2 and CFX-3 complement the given 2's complement representation of fixed-point operand in register UQ so that a positive mantissa can be obtained. Then the control branches to task stage CFD-3 and follows the same path as in case of decimal operands.

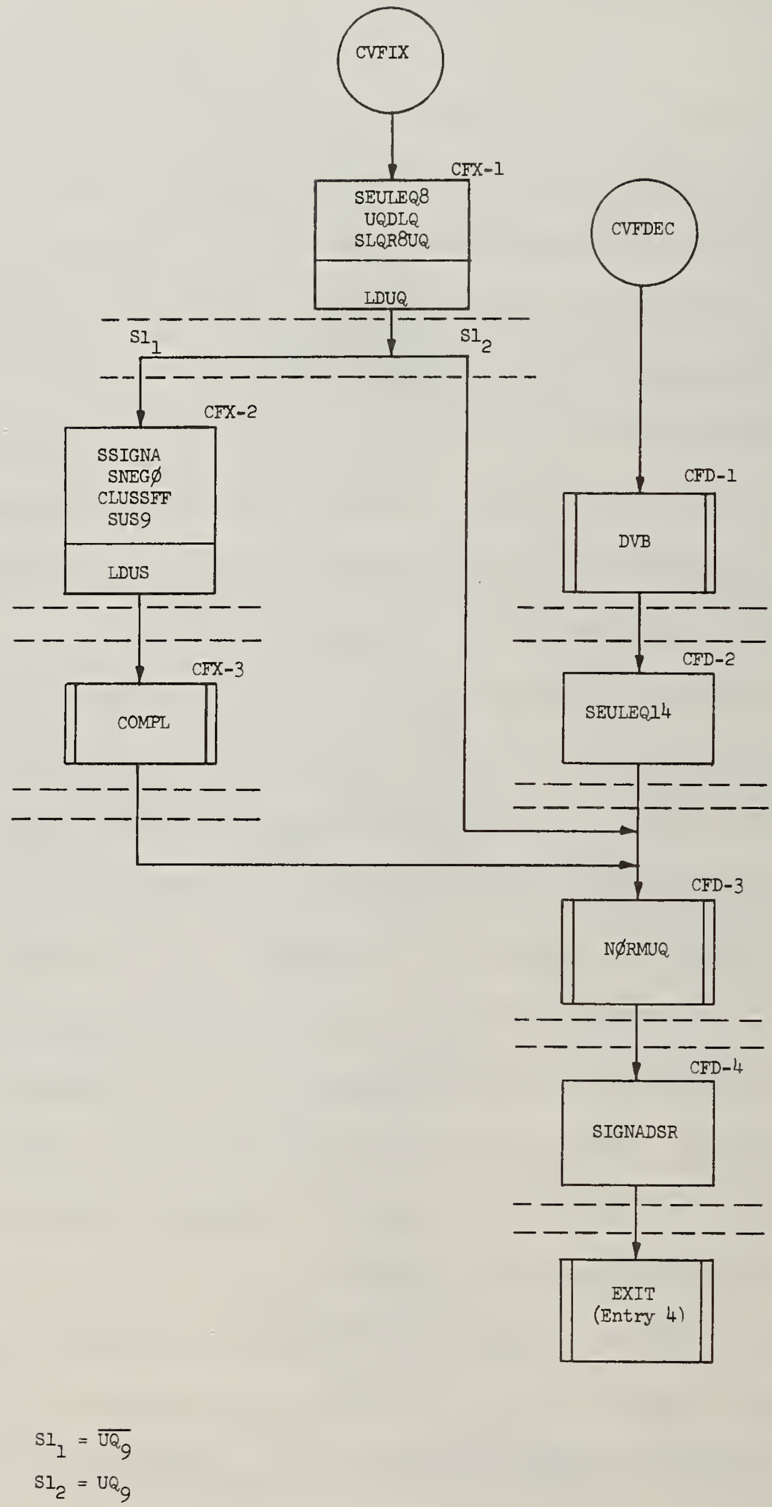


Figure 3.8.3.2.1. CVF_DEFIX—CVF Decimal and Fixed Point Operands

3.8.3.3 Logic Implementation

In the table below is shown the names of various control subsequences which make up the CVF control sequence, the figure numbers of their flow charts and the number of the corresponding logic drawing which implements the control sequence in hardware.

Control Subsequence name	CP Flowchart Figure No.	Corres. Logic Drawing No.
DVF-DEFIX	3.8.3.2.1	AUO-07-383-01
DVB	3.8.2.2.3.2 3.8.2.2.3.3	-382-04 -382-05
COMPL	3.8.2.2.5.1	-382-05
NØRMUQ	3.4.2.1.1	-352-04
REFOTRAN		-332-05 352-04 -05 -06

3.8.4 CONVERSION TO DECIMAL (CVD) CONTROL SEQUENCE

This control sequence is used to convert either a fixed point number (both short and long) or a floating point number into a decimal number of Illiac III format.

3.8.4.1 Global Flow Description

Figure 3.8.4.1.1 shows the global flow diagram for CVD control sequence. This control sequence comprises of many subsequences some of which are shared by other arithmetic order control sequences.

The first subsequence CVD-FIXOCAFLOX calls upon the subroutine control sequence COMPL to complement the given fixed point number if it is negative. In case of floating point number, a check is made to see if the given floating point number will give rise to an out-of-range converted decimal number. If so, the control flip-flop ØV is set. Further, if the exponent is so large that no significant figures in the converted number can be represented in result register UQ of finite length, then the result register is cleared and the control branches to the procedure control subsequence SETSIGN which generates the decimal code for sign of the number and finally the control enters the terminal sequence REFOTRAN.

If, however, the converted number is within the range of representation with at least some significant digits, the control calls upon the subroutine control subsequence FL-NØRM-FX which converts the floating point number into an equivalent fixed point representation.

The control then branches to subsequence GETDEC which converts the so far obtained equivalent fixed point number into an equivalent decimal number, in concert with the control subsequence QUODECM. The control sequence GETDEC calls upon the subroutine control subsequence FORMQAREM at entry marked CVDIV to divide contents of UQ or partial remainders by 10 to obtain decimal digits. Subsequence QUODECM decreases quotient by 1, if the remainder obtained by FORMQAREM is negative.

After the equivalent decimal number has been obtained, the control subsequence GETDIG transfers the converted decimal number to the result output register UQ and then the control passes on to the control subsequence SETSIGN. This subsequence generates the decimal code for sign of the result in the result register and finally the control enters the terminal sequence REFOTRAN for flag setting and final onward transmission to the TP.

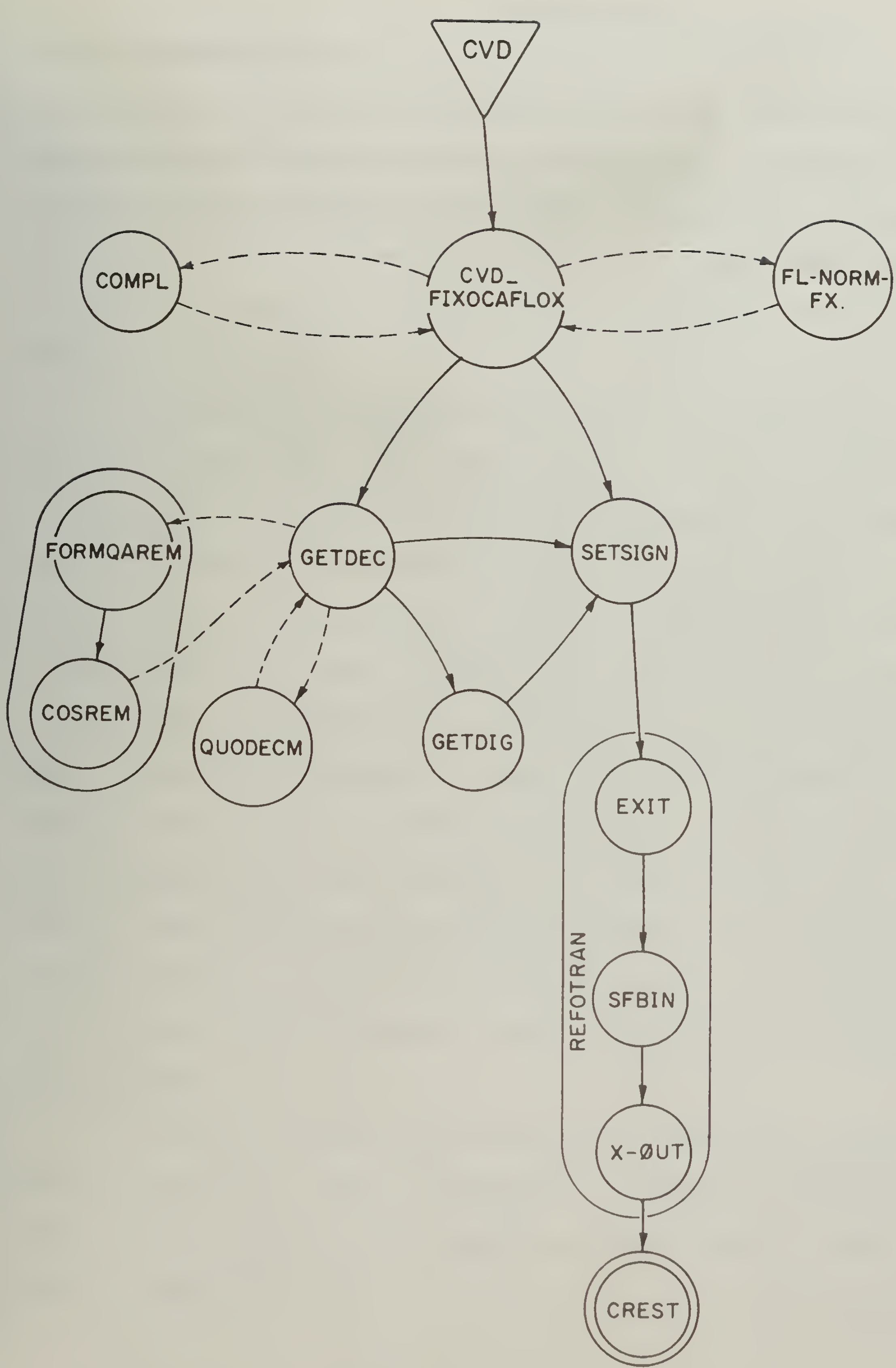


Figure 3.8.4.1.1. Global Flow Diagram for "CVD--Conversion to Decimal" Control Sequence

3.8.4.2 Control Point Flow Description

The control point flow chart of CVF control sequence involves many control point subsequences as explained in the global flow diagram. A brief description of control point flow chart associated with each of the subsequences is given below. These flow charts resemble very closely the simulation flow charts.

3.8.4.2.1 CVD-FIXOCAFLOX

Figure 3.8.4.2.1 shows the control point flow chart for this subsequence. For a fixed point number, the number to be converted is in register UQ bits 33 through 64, where bit 33 is the sign bit. For a floating point operand, the mantissa of number to be converted is in register UQ bits 9 through 64 and the exponent is in register EUU of the exponent arithmetic unit.

Task stage CVD-1 provides control signals to transfer the exponent via the EU Adder to the input of register EUL of the exponent arithmetic unit. Note that this task stage is entered even if the given number to be converted is of fixed point type. It is unnecessary to do so, but it was allowed to stay because it does not do any harm. Task stage CVD-1 also temporarily stores the sign of the operand in control flip-flop SR.

If the fixed-point number to be converted is negative ($S1_1$), task stage CVD-2 transfers the number to register US and sets the control signal NEGØ to appropriate state so that the number can be complemented by subroutine control sequence COMPL which is called by the task stage CVD-3. After the number has been complemented, the control branches to procedure control subsequence GETDEC. If number to be converted is fixed point and positive ($S1_2$), the control from the sequence stage S1 directly passes to the subsequence GETDEC.

In case of floating point operand ($S1_3$), task stage CVD-4 gates the output of exponent unit adder to register EUL. Transfer of exponent to register EUL is necessary because exponent magnitude 'detection logic' is at the output of register EUL only.

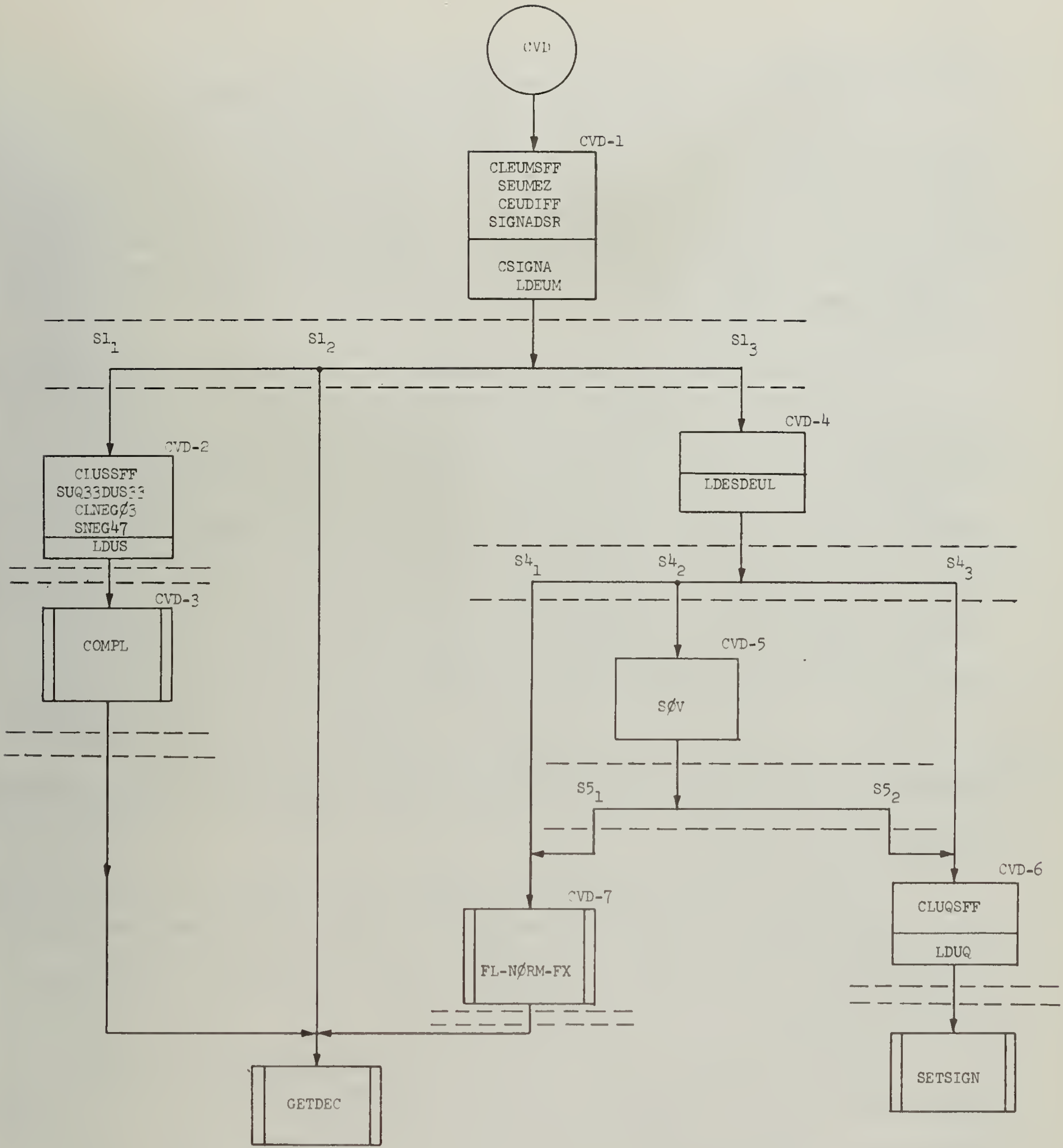
Since the decimal number can have only 14 digits and if the absolute value of the floating number to be converted is greater than $(10^{14}-1)$ which is less than 16^{12}, an overflow indicator should be set if exponent is ≥ 13. Task stage CVD-5 sets the control flip-flop ØV when such a condition occurs. However, if the $13 \leq$ exponent ≤ 28 (sequence condition $S5_1$), the converted number will still have some significant digits and so the control branches to task stage CVD-7 which calls the subroutine control subsequence FL-NØRM-FX. If exponent >28, the converted number is too large to be represented by any significant digit in the finite length result register UQ and the control goes to task stage CVD-6 which clears the result register UQ. Then the control branches to procedure control subsequence SETSIGN to generate the decimal code for sign of the number.

In sequence stage S4, if 0 < exponent < 13 as indicated by condition $S4_1$, the control directly goes to task stage CVD-7. However, if the exponent ≤ 0, the given floating point number is a fraction and hence the equivalent decimal integer number is zero and so the control branches to task stage CVD-6.

Task stage CVD-7 calls the subroutine control subsequence FL-NØRM-FX which transforms the given floating point number to a fixed point number.

From the output of task stage CVD-7, the control passes on to the procedure control subsequence GETDEC.

The reader should note that the simulation flow chart on page 106 of DCS Report No. 418 does not show the checking of condition EULGE13 to set the overflow indicator. But, it still would not cause any problems because the overflow condition would again be detected in procedure GETDEC when ND becomes greater than 14.



$S1_1$ = FIX.UQ33 $\quad$ $S1_3$ = FLT $\quad$ $S4_2 = \overline{\text{DEXPGE13}}$ $\quad$ $S5_1 = \overline{\text{EULGT28}}$

$S1_2 = \text{FIX.}\overline{\text{UQ33}}$ $\quad$ $S4_1 = \overline{\text{DEXPGTZ}}\cdot\overline{\text{DEXPGT13}}$ $\quad$ $S4_3 = \overline{\text{DEXPGTZ}}$ $\quad$ $S5_2$ = EULGT28

Figure 3.8.4.2.1. CVD_FIXOCAFLOX—CVD_Fixed Point Operands' Complementation and Floating Point to Fixed Point Conversion

3.8.4.2.2 GETDEC

Figures 3.8.4.2.2.1 and 3.8.4.2.2.2 show the flow charts for control subsequence GETDEC which provides control signals to generate decimal equivalent of the contents of UQ. The decimal digits are available in register M at the end of the sequence. In brief, the algorithm to do the conversion is as follows.

The contents of register UQ is used as dividend and is divided by 10. The decimal digits are generated from the least significant to the most significant. The remainder after each division is a decimal digit. The M register is used to store the converted decimal digit. Every time a division is completed, the contents of M is right shifted four bits and the new remainder (bits 9-12 of LM register) i.e. the decimal digit is inserted in bits 9-12 of M. The quotient formed in registers UH and UQ is used as the new dividend to calculate the next decimal digit and the whole process is repeated until the number of decimal digits formed exceeds 14 or the contents of register UQ is zero.

Task stage CVD-8 loads the division loop counter ICC and also clears the decimal digit counter ND. The control now passes to decimal-digit-calculation loop.

Task stage CVD-9 increments the counter ND. Sequence stage S9 checks if more than 14 digits have already been formed, and if so, the decimal number has overflowed. Thus, the control branches to task stage CVD-10 which sets the control flip-flop ØV and later the control goes to procedure control subsequence SETSIGN. If, however, the number of decimal digits formed so far has not exceeded 14, more decimal digits need to be formed and the control goes on to activate,in parallel,the task stage CVD-11 and the dividend-scaling loop consisting of task stages CVD-13, 14, 15.

A difference from the simulation flow chart should be noted here. In simulation manual, the shifting of M by four bits to the right is done first, and the scaling of dividend in register UQ, UH is done later in time sequence due to sequential nature of simulation. However, since the control hardware can execute in parallel, the M-shift control consisting of task stages CVD-11 and CVD-12 and the dividend scaling loop are activated in parallel because both involve separate and independent hardware.

Task stage CVD-11 sets up the SDS-array and sets the control signal MDY4 to transfer contents of M to the output of SDS array. At the same time, it sets up the data path for the right shift of the contents of register LM and then activates a delayed gate signal LDM to load the contents of LM right shifted four bits into register M. The control then waits for the dividend-scaling loop to finish.

Dividend scaling loop scales the dividend in register UQ lies between 1/2 and 1 and thus decreases the number of passes through the division loop. Task stage CVD-13 shifts the contents of UQ and UH, left by eight bits and at the same time sets the ICC counter to downward counting state. Task stage CVD-14 activates the count down pulse and the control loops back to do further shifting. If all the top eight bits of the dividend are not zero, the contents of UQ and UH are shifted left by four bits and then the control waits for a reply from the task stage CVD-12 of M-shift logic.

When both the in-parallel activated control chains have replied as indicated by the 'JOIN' symbol, the task stage CVD-16 calls the subroutine control sequence FORMQAREM at entry CVDIV to divide the dividend in UQ by 10.

The subsequence FØRMQAREM gives rise to unassimilated quotient in registers UQ, UH and the assimilated remainder in register LM. If, however, the remainder is negative, then subsequence COSREM corrects the remainder such that it becomes positive and also sets the flip-flop NEGR. This corrected remainder is the correct decimal digit.

The control now passes to procedure control subsequence QUODECM whose control point flow chart is shown in Figure 3.8.4.2.2.2. Sequence stage S16 tests whether the control flip-flop NEGR is set. If so, the remainder in subsequence FØRMQAREM was negative and the quotient must be decremented by unity. Task stages CVD-17, 18 and 19 perform this function via the SDS-array.

Task stage CVD-17 transfers the unassimilated quotient from registers UQ, UH to the registers UM, US which serve as input SDS-array.

Task stage CVD-18 sets up and activates the SDS-array for decrementing the contents of US, UM by unity. Task stage CVD-19 is a dummy stage and only acts as a time buffer to allow the SDS-array to stabilize.

Task stage CVD-20 transfers the bits 9-12 of LM to the bits 9-12 of register M. This is the corrected remainder (hence, the correct decimal digit) obtained earlier in subsequence COSREM of subsequence FØRMAQAREM.

It should be carefully noted here that whereas in the simulation flow chart the transfer of LM_{9-12} to M_{9-12} is shown to be done before decrementing the quotient, it cannot be done in actual hardware. The reason is that in task stage CVD-18, one needs to clear the register M and turn on control signal SETM64ØNE to achieve $Y_{1-63} = 0$ and $Y_{64} = 1$ for unity decrementation of the quotient. So it had to be postponed to the task stage CVD-20 but before CVD-21 because in CVD-21 the original contents of LM is destroyed.

Task stage CVD-21 transfers the decremented quotient from the output of SDS-array to registers LS, LM and then gates them into the registers UH, UQ, respectively.

Task stage CVD-22 again transfers this corrected quotient which now serves as the new dividend for the calculation of another decimal digit to input registers US, UM of SDS-array and the control loops back again to calculate new decimal digit.

The control continues to loop till either the quotient is zero ($S8_2$) whence the control branches to subsequence GETDIG or till 14 decimal digits have been formed in which case an overflow flip-flop ØV is set by task stage CVD-10 and control branches to SETSIGN control subsequence.

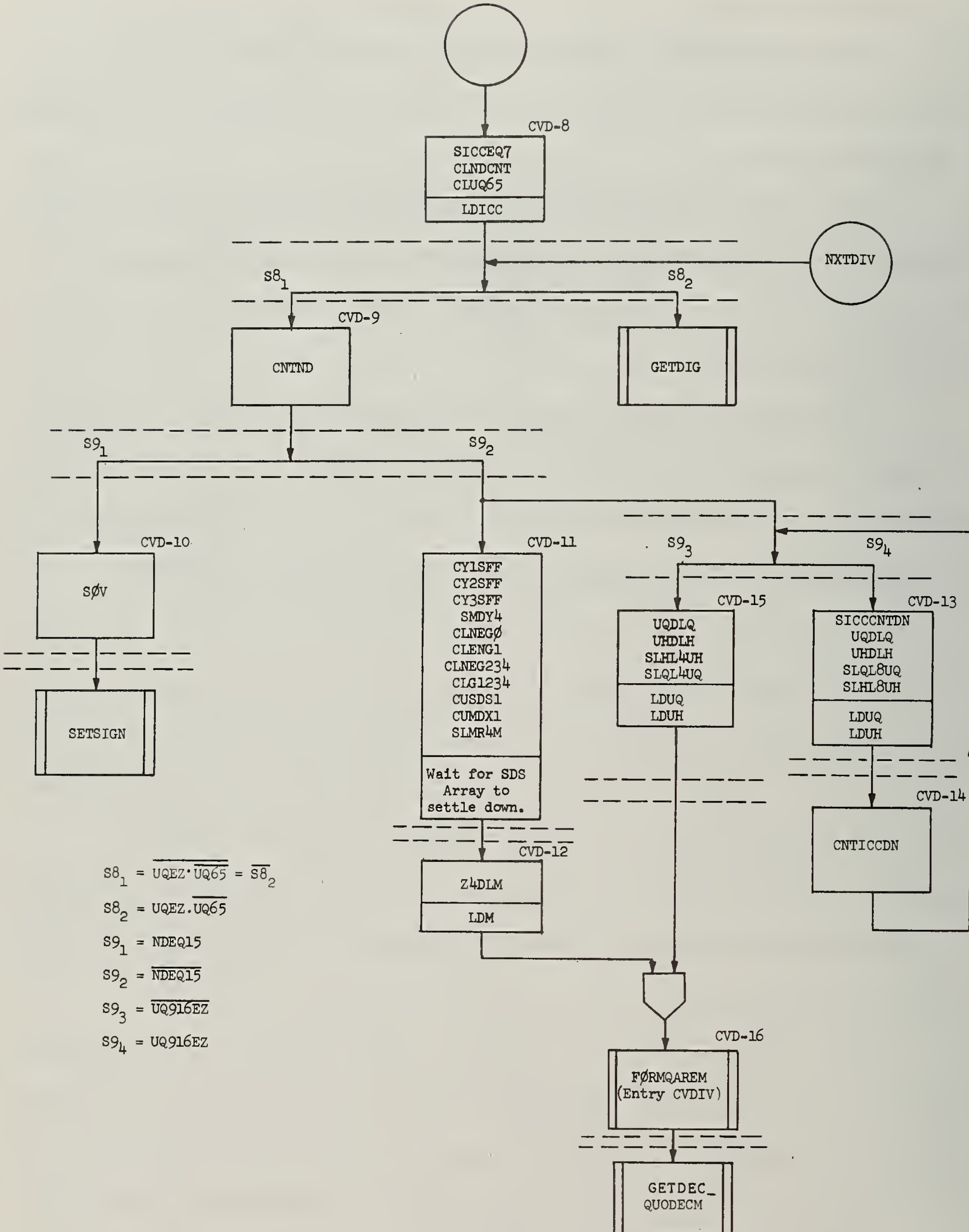


Figure 3.8.4.2.2.1. GETDEC_CAQOREM—Generate Decimal Digits—Calculation of Quotient and Remainder

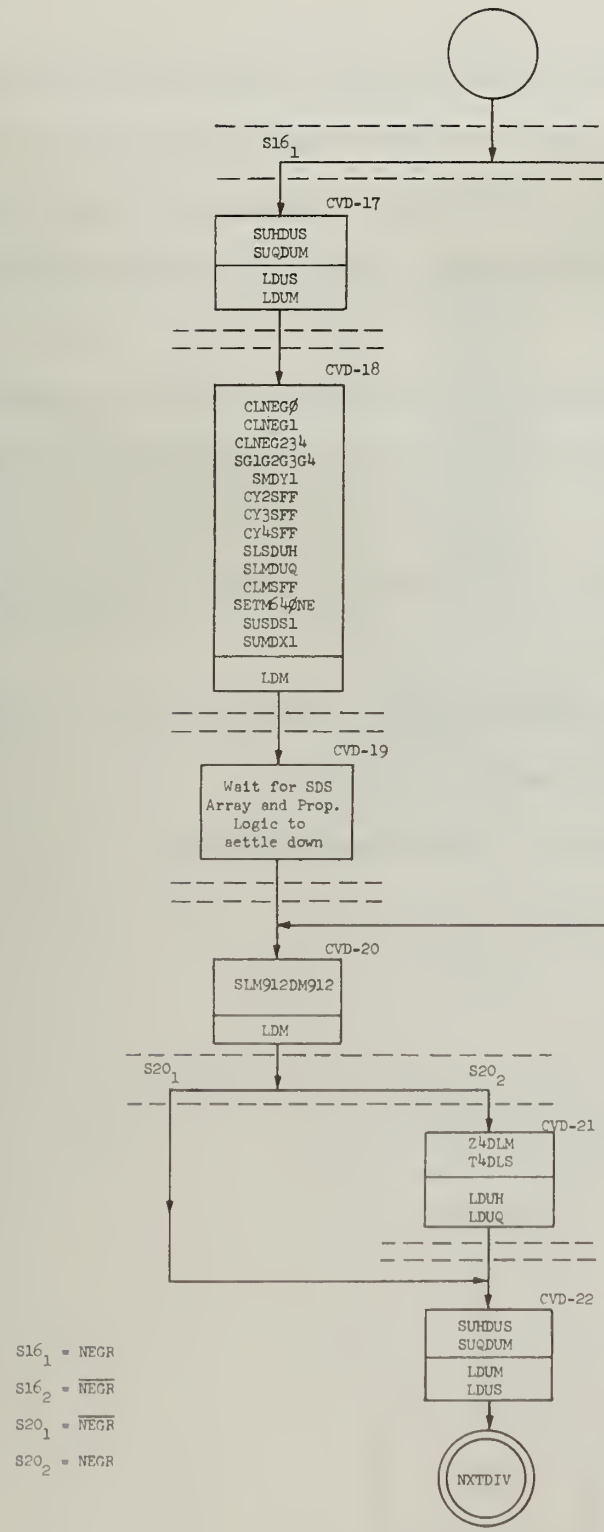


Figure 3.8.4.2.2.2. GETDEC_QUODECM--Generate Decimal Digits--Quotient Decrement

3.8.4.2.3 GETDIG

Figure 3.8.4.2.3.1 shows the control point flow chart of control subsequence GETDIG, whose function is to transfer the converted decimal number in register M to the output result register UQ. Since there is no direct path from M to UQ, the transfer takes place through the SDS-array and registers LM, LS.

Task stage CVD-23 sets up the SDS-array for the transfer of the contents of M to the output of SDS-array via the M-shift array control signal MDY4.

Task stage CVD-24 loads the output of SDS-array into register LM and clears the register UQ.

Task stage CVD-25 transfers the contents of LM, the converted decimal number to the output result register UQ.

Then the control goes to subsequence SETSIGN.

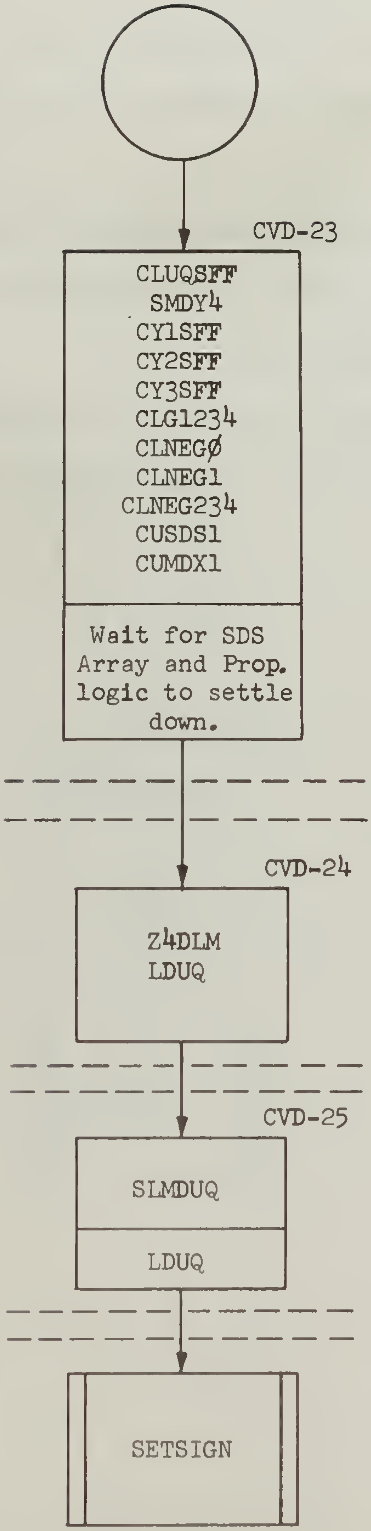


Figure 3.8.4.2.3.1. GETDIG—Transfer Decimal Digits to Result Register UQ

3.8.4.2.4 SETSIGN

Figure 3.8.4.2.4.1 shows the control point flow chart of control subsequence SETSIGN. This subsequence is very short and involves only one task stage.

Task stage CVD-26 generates the binary pattern for the decimal code for plus or minus sign of the number to be converted and gates it into bits 5-8 of result register UQ.

Now the result is ready to be transferred out to the processor and the control passes on to the terminal sequence REFOTRAN via the subsequence EXIT.

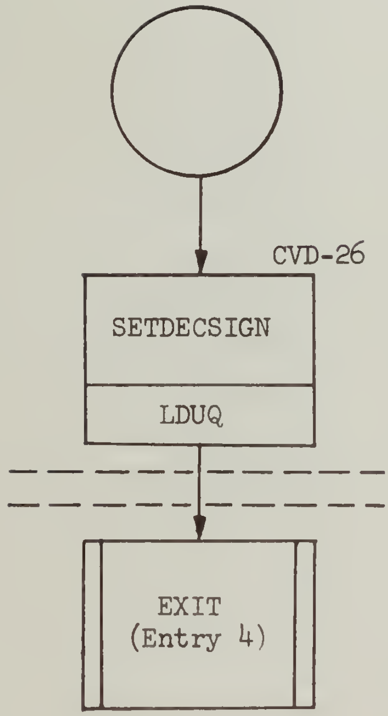


Figure 3.8.4.2.4.1. SETSIGN—Set Proper Decimal Sign Code in Result Register UQ

3.8.4.3 Logic Implementation

Table below shows the names of the various control subsequences which comprise the CVD control sequence. It also lists the figure number of their flow control point flow charts and the number of corresponding logic drawing which implements the subsequence in hardware.

Control Sequence Name	CP Flowchart Figure No.	Corres. Logic Drawing No.
CVD_FIXOCAFLOX	3.8.4.2.1	AUO-07-384-01
FL-NØRM-FX	3.8.2.2.2.1	-382-03
COMPL	3.8.2.2.5.1	-382-05
GETDEC_CAQOREM	3.8.4.2.2.1	-384-02, 03
FØRMQAREM	3.7.2.3.3	-371-08
COSREM	3.7.2.3.4	-371-09A
GETDEC_QUODECM	3.8.4.2.2.2	-384-04
GETDIG	3.8.4.2.3.1	-384-05
SETSIGN	3.8.4.2.4.1	-384-05
REFOTRAN		-332-05 -352-04 -05 -06
Task Signal Collector "D"		-384-06

3.9 INITIALIZATION AND POWER TURN-ON.

This section describes the initializing control subsequence CREST which forms the last part of every arithmetic control sequence. This sequence clears all the memory elements in the control logic and readies the control for the processing of next arithmetical order. In addition, this section also describes very briefly the power turn-on.

3.9.1 CREST

Figure 3.9.1.1 shows the control point flow chart of this control sub-sequence. There is hardly any sequence involved here because it consists of only one task stage. This sub-sequence is entered at the end of the control sequence of every arithmetic order after the terminal sequence REFQTRAN. This sub-sequence is used to clear every memory element in the control hardware e.g. control points, control flip-flops, selector switch flip-flops as well as many registers in the processing hardware.

Task stage CLR-1 is implemented with a DETAG type control point, as shown in Drawing AUO-07-399-01. The regular task signal activates the control signal COMCLR which clears most memory elements except the hardware registers. Control signal COMCLR puts an all-zero pattern on the input of various hardware registers by resetting all the selector switches, but a gate signal is necessary to feed this all-zero pattern into the registers to clear them. This signal is provided by delayed task signal CLREGS (CLR1DT1) which acts as the clearing gate signal to the various registers.

After this clearing and resetting operation is over, the control goes back to Operand Input load (VIN) control sequence to await the arrival of a new arithmetic order and operands for processing by AU.

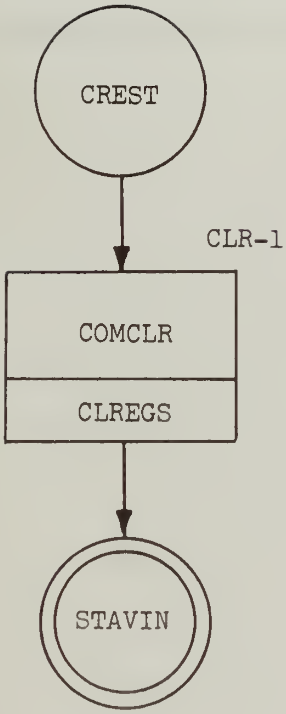


Figure 3.9.1.1 - CREST - Clear and Reset

3.9.2 Power Turn-ON

Turning on the power or depressing a pushbutton labelled CL (Clear) in the AU activates the task stage (hence the control point) CLR1 of the control sequence CREST. This subsequence clears (resets) every memory element in the control logic hardware, for example, control points, control flip-flops selector switch flip-flops in addition to many registers in the Processing hardware logic. Section 3.9.1 gives an operational description of control sub-sequence CREST.

The logic implementation of power turn-on mechanism is shown in Drawing AU0-07-399-01.

3.10 'MISCELLANEOUS' CONTROL LOGIC

This section briefly describes various segments of control logic hardware which are unrelated to each other. These segments of control logic do not involve any control point flow charts. In this section, the mechanisms for electronic push-button selection of data paths and for single stepping through the control sequence are discussed. In addition, the final level task signal drivers' logic is also explained

3.10.1 Electronic Push-button Selector Switches

There are many registers in the arithmetic unit which have inputs from many different sources. At any time, in general, the registers receive input from only one source. So it is necessary that at any one time, only one input path be active and the others should be inhibited. Electronic push-button selector switches provide such a function very conveniently. These switches are shown in Drawings AUO-07-390-01 to AUO-07-390-08.

Basically, these selector switches consist of as many flip-flops as there are paths at the input of a register. The output of each flip-flop activates one unique path. The set and reset logic at the input of each flip-flop is very simple and acts in such a way that the set signal for one flip-flop acts as the reset signal for the other flip-flops. Thus, at one time only one flip-flop is active which activates the desired path and the other paths are inhibited because the other flip-flops are reset. That is why, this selector switching logic is called the Electronic Push-button selector switches. It is called electronic because flip-flops are electronic and no mechanical switches as such are involved. The various registers whose input selector paths are controlled by electronic push-button selector switches are UM, US, UQ, UH and register M. Besides the above registers, electronic selector switches also control SDS-array's gate signal Gi and negation signal NEGi and the control signals of M-shift array and Model Division's Dividend selector signals.

3.10.2 Single Step Mode

The single step mode permits one to manually control the sequential turn-on and turn-off of each control point (associated with an asynchronous operation) in any control sequence. The implication in this statement is that stepping is NOT permitted in those control points which are used for transferring data into or out of the AU.

The two signals labelled EN (Enable) and GODELY (Godelay) shown in Figure 3.10.2.1 control the stepping operations. These two signals are tied to corresponding inputs of all Control Points associated with asynchronous sequencing operations. Both of these signals are in logical '1' state when the machine is in the Run Mode i.e. not in the Single Step Mode. When in the Signal Step mode, these two signals are always complementary (EN $\neq$ GODELY) of each others. Note that when EN = 0, a Control Point may be "primed" (i.e. its flip-flop is set) but cannot activate a task signal; when GODELY = 0, a Control Point can come on (i.e. activate a task signal) but it cannot be turned off (i.e. its flip-flop cannot be reset). These features together with an inherent property of the Control Point. (The coming-on of any Control Point i primes the next control Point i + 1 in the sequence, but does not permit Control Point i + 1 to come on. Control Point i + 1 is only permitted to come on after Control Point i has gone off) permit the stepping mode operation.

The Single Step Mode involves the use of three switches shown in the Drawing AUO-07-399-01. These switches are called the Stepping Mode Switch SM, the STEP/RUN switch SR and the Step Switch ST.

The SR switch has two positions - STEP and RUN. In the RUN position, EN = GODELY = '1'; in the STEP position, the ST switch determines which of the logical conditions "EN = '0', GODELY = '1'" or "EN = '1', GODELY = '0'" occurs.

The SM switch provides the option of stepping by single instruction. This is accomplished by putting the SM switch in the Single Instruction Mode (SINØD) position. In this position, the alternating EN and GODELY signals are only applied to Control Point CLR1; for all other Control Points, EN = GODELY = '1'. When the switch SM is not in the SIMØD position, the alternating EN and GODELY signals are applied to all Control Points, including CLR1.

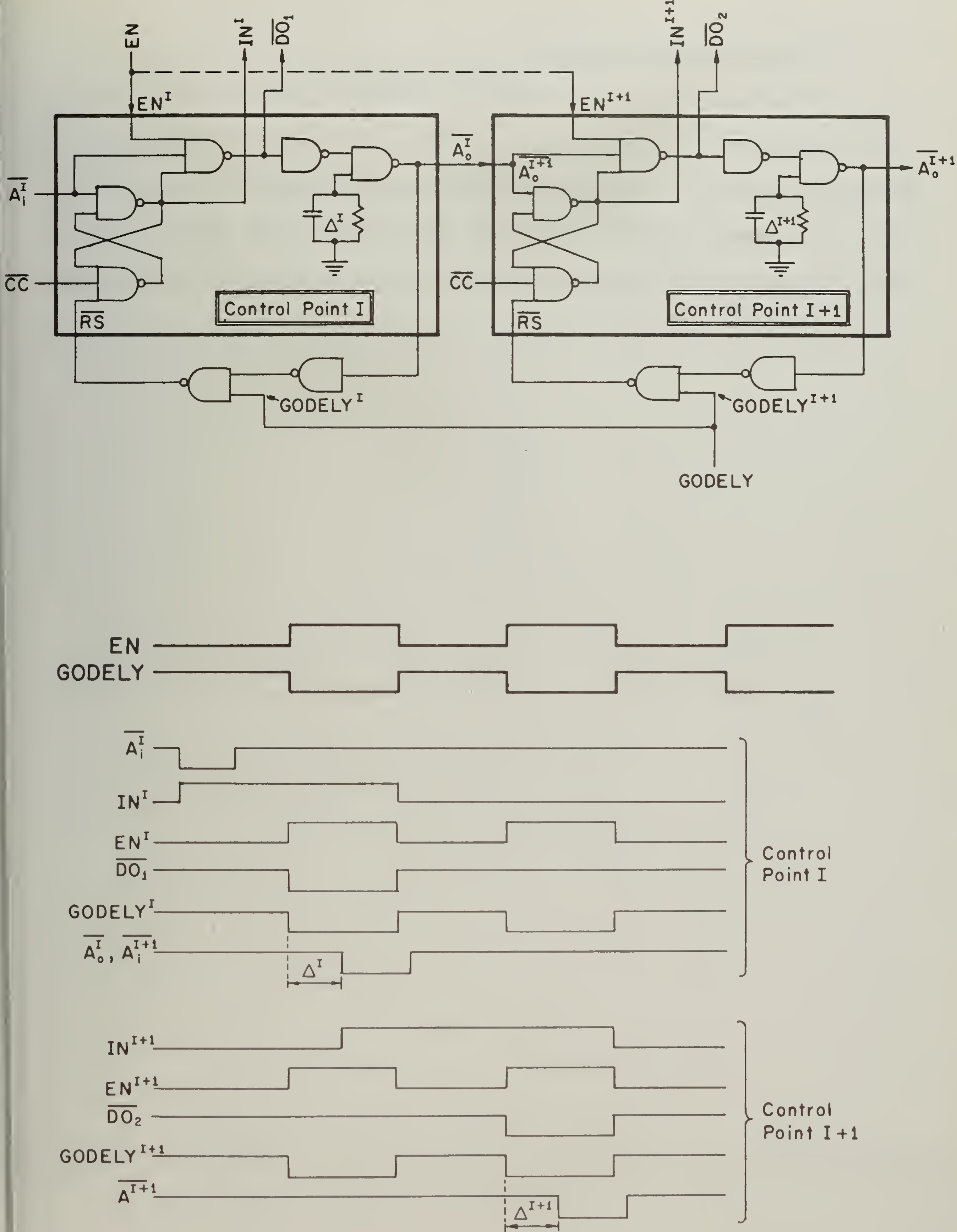


Figure 3.10.2.1. Illustration of Single Step Mode

3.10.3 Final Level Task Drivers

This task driver logic represents the final level of task signals' collection before these task signals are taken to the input of processing hardware through the interface logic. The implementation of this logic is shown in logic Drawings AU0-07-388-01 to AU0-07-388-03. The input to these drivers is in general the output from the task signal collector logic of each arithmetical control sequence.

4. AU DRIVER INTERFACE

4.1 Introduction

There is need for a driver interface between the control logic hardware and the processing hardware because of the load requirements and the logic level differences. This section describes briefly the interface for signals that run from control hardware to processing hardware and vice versa.

4.2 Control Logic to Processing Hardware Driver Interface

A large number of control signals generated in the control hardware go to activate the various functions in the processing hardware and each control signal may involve, in general, quite a heavy load on it. Secondly, the output of TTL logic with which control hardware is implemented is different from the level required at the input of DTL logic from which the processing hardware is constructed. Hence a driver interface has been provided to transform the logic levels appropriately and also to provide the necessary drive capability. DTL logic card 1018-262-XX provides the necessary drive and the level transformation and acts as the interface logic card. This interface logic is shown in control Drawings AUO-07-400-04 to AUO-07-400-12.

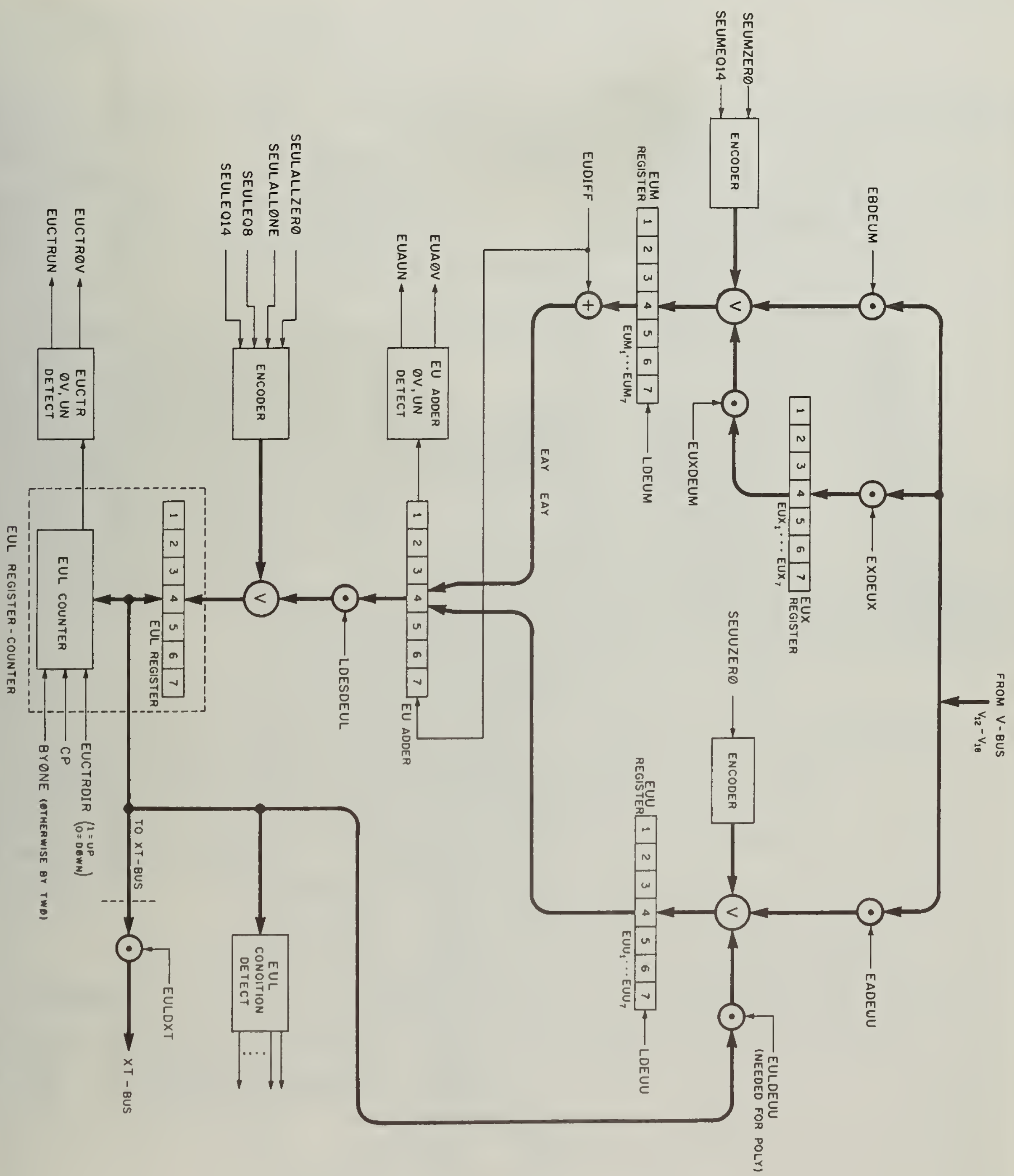
4.3 Processing Hardware to Control Logic Driver Interface

There is a relatively small number (~50) of signals which originate in the Processing hardware and terminate in the Control logic hardware. The drivers for these signals are not collected and shown on a separate set of drawings as in the case of Control to Processing hardware driver interface. Rather they are shown on the various Processing hardware drawings themselves where the signals originate. Spare drivers left over from the unused sections of Logic cards in Processing hardware are used as interface drivers. These drivers more than meet the current requirements of the load but fail to meet the voltage (level) requirements. To make these drivers compatible with the load, terminating resistors are added to the load. For more detailed information concerning levels, loading, hardware description and other notations etc., the reader should consult DCL File #872 written by Mr. Paul Krabbe.

5.0 CONTROL LOGIC DRAWINGS AND THEIR INDEX

This section contains an index of Control Logic drawings which show the logic implementation of various control sequences and sub-sequences discussed earlier. The index shows the drawing number followed by the title of the drawing and the corresponding page number on which the drawing appears.

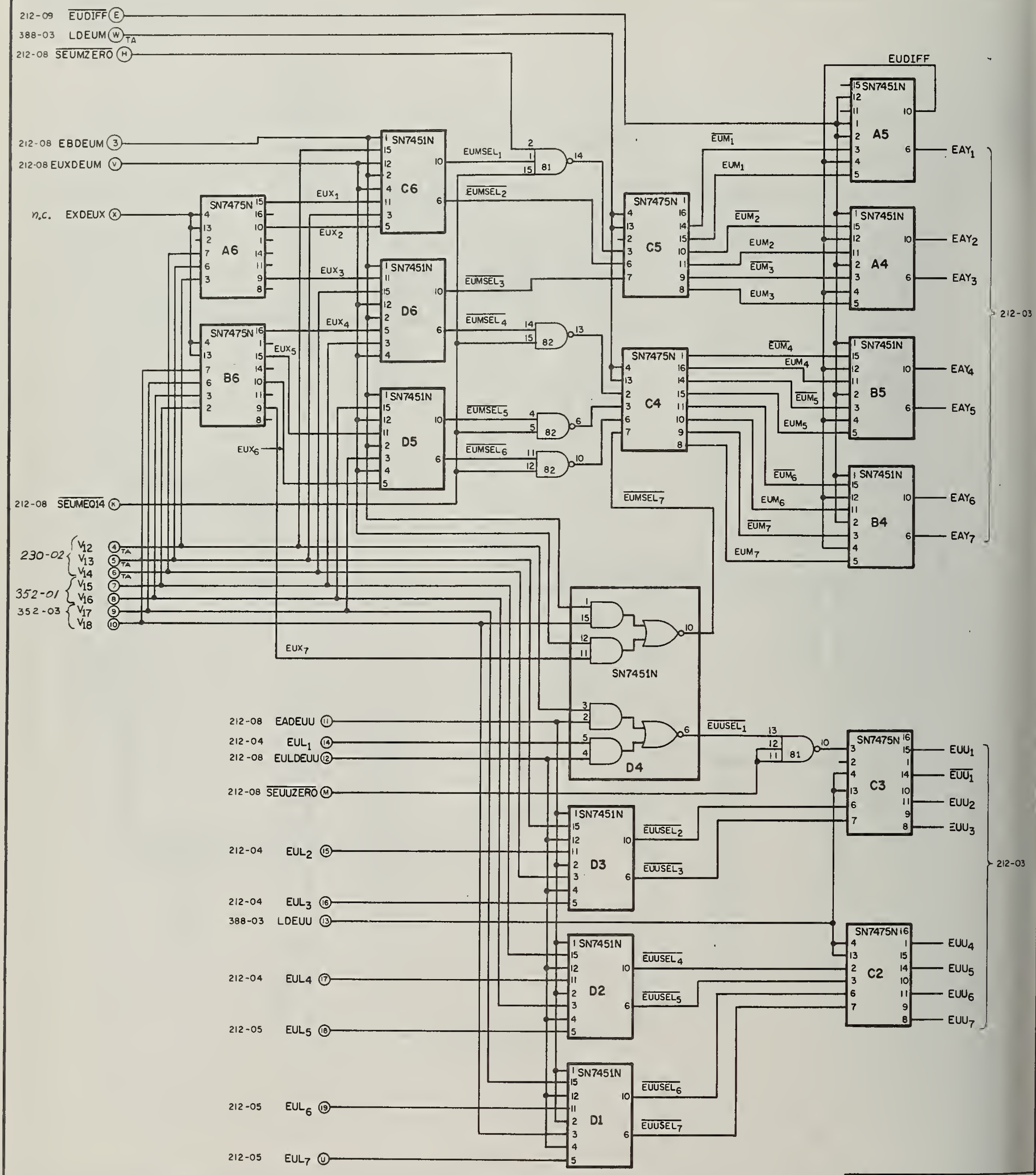
The index is followed by a set of the Control Logic drawings. The Control Logic drawings reflect the latest state of the logic, as it appears at the time of writing of this report. These drawings may undergo some minor changes and modifications due to shortcomings which will become known in the check-out phase.



Issue	Change order	Approved by	Date

REVISIONS

DEPARTMENT of COMPUTER SCIENCE University of Illinois				TITLE AU EXPONENT - ARITHMETIC LOGIC BLOCK DIAGRAM
For L.G	Drawn by S Z	Date 1-28-72	Supersedes dwg.	
Approved by	Date 1 28 1972	Reference		Sheet ____ of ____ Drawing No. AUO-07-212-01



CARD LOCATION: C-26

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

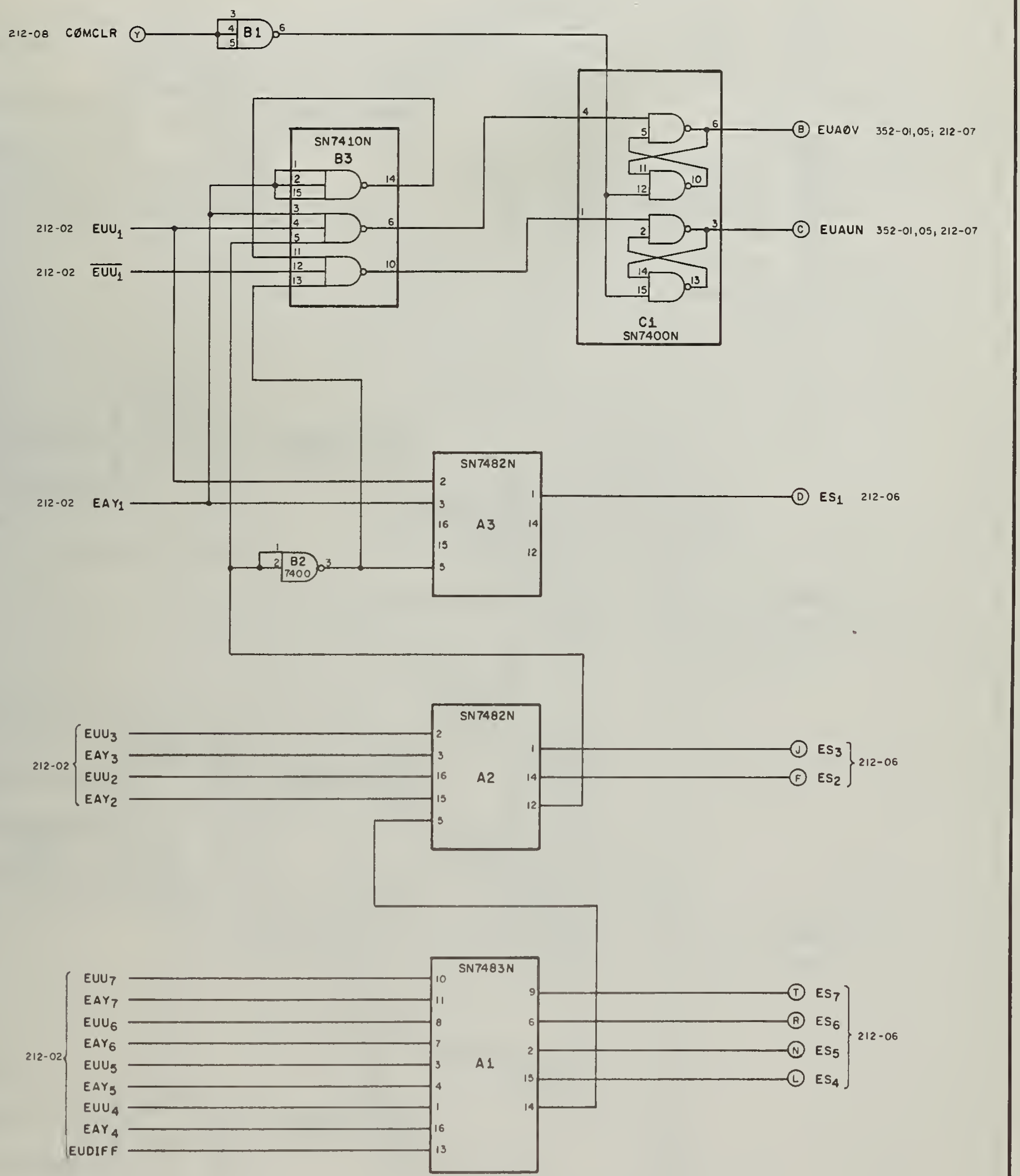
TITLE AU EXPONENT - ARITHMETIC LOGIC EUM AND EUU REGISTERS WITH IN/OUT GATING

For	Drawn by	Date	Supersedes dwg.
LNG	LRS	5-26-71	
Approved by	Date	Reference	
RM	6-15-71		

Sheet ____ of ____ Drawing No. AU0-07-212-02

Issue	Change order	Approved by	Date
2	No. 528	P. Krabbe	4-12-72

REVISIONS



CARD LOCATION:

C-26

2	No 529	P Khalla	4-6-72
Issue	Change order	Approved by	Date

REVISIONS

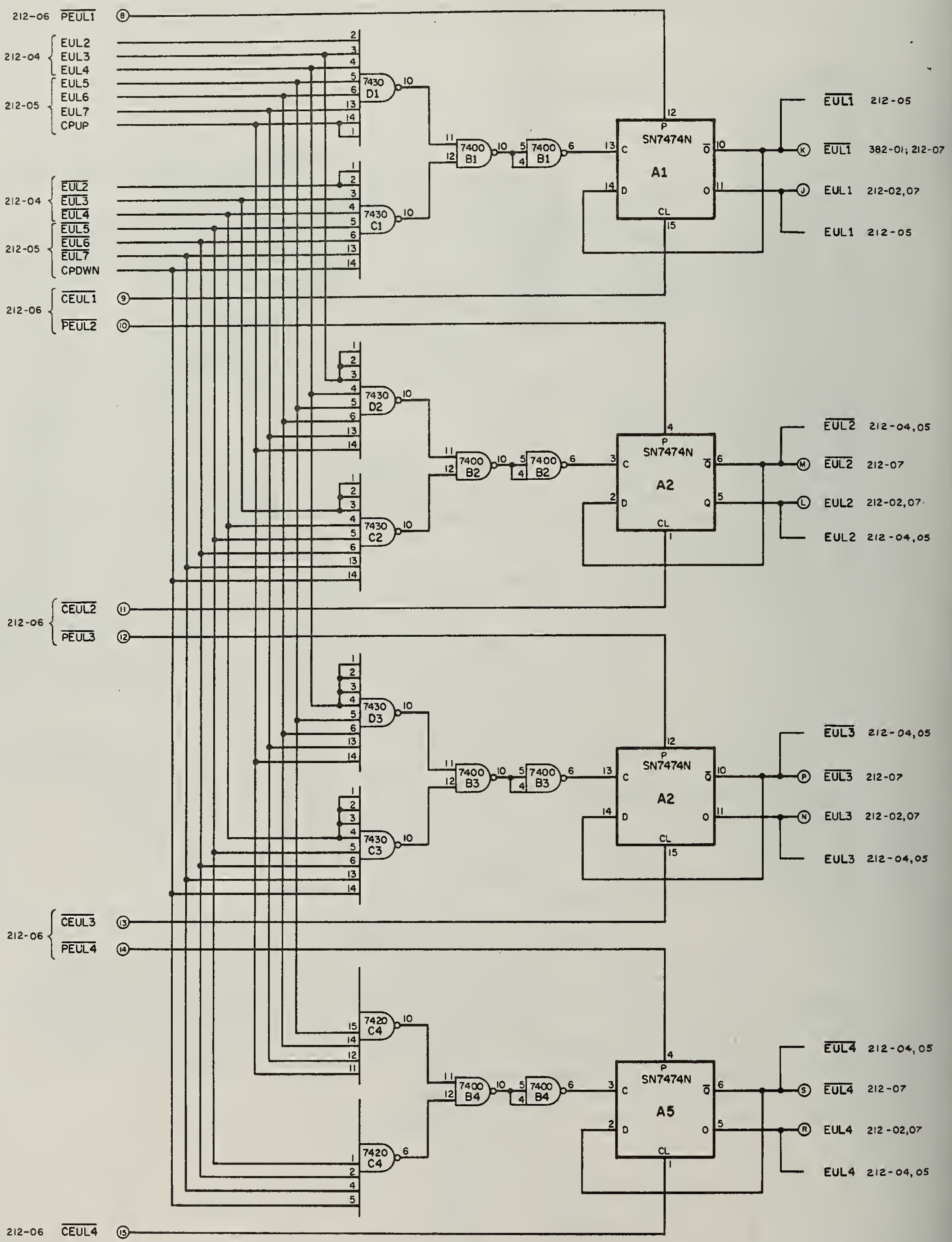
DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L G	S Z & T G	7-6-71	
Approved by	Date	Reference	
[signature]	8-7-1971		

TITLE AU EXPONENT - ARITHMETIC LOGIC
EXPONENT UNIT ADDER

Sheet _____ of _____

Drawing No. AUO-07-212-03



CARD LOCATION:

C-22

Issue	Change order	Approved by	Date
2	No. 530	P. Krabbe	4-12-72

REVISIONS

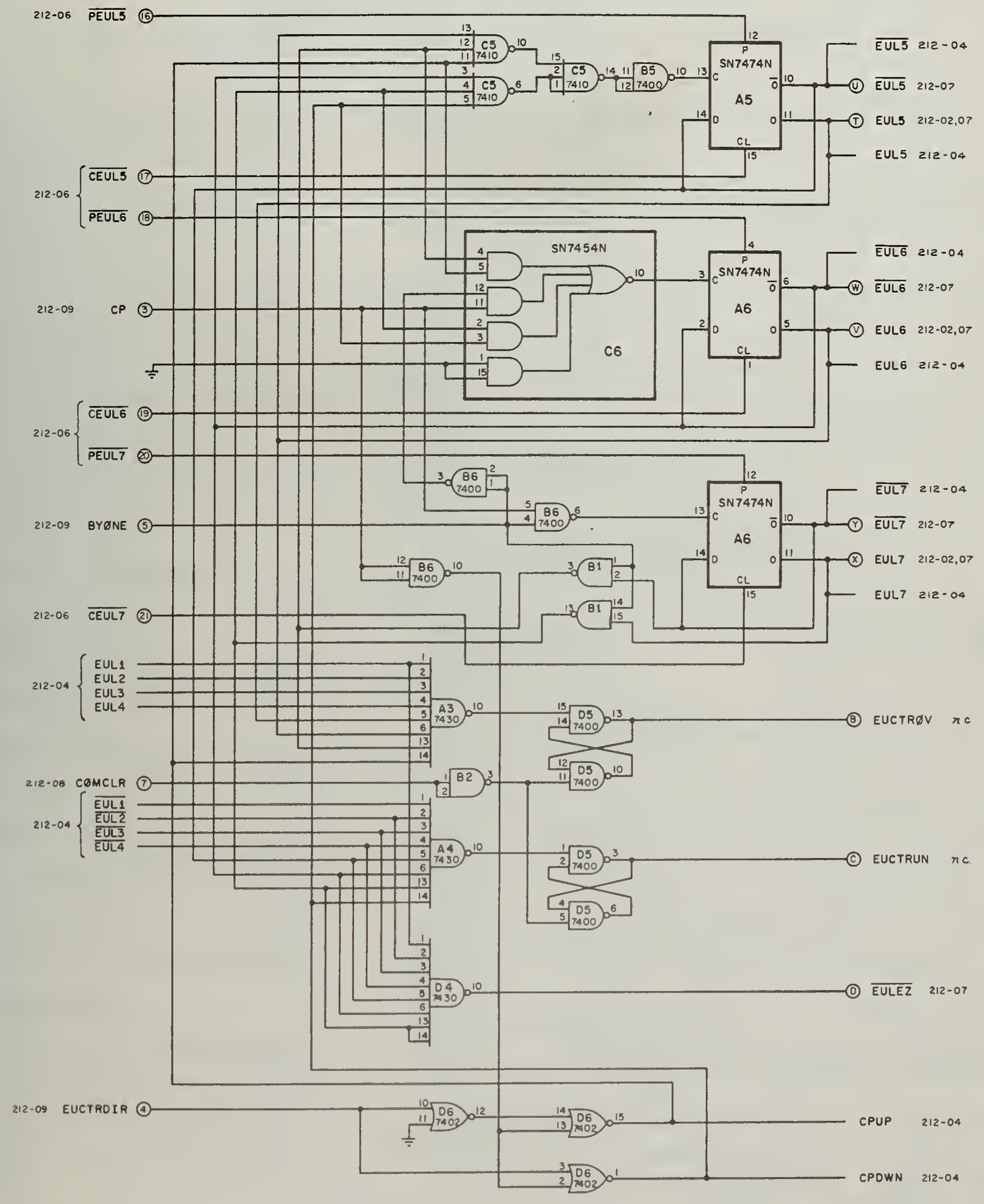
DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
LNG	LRS	5-20-71	
Approved by	Date	Reference	
[illegible]	6-15-71		

TITLE AU EXPONENT-ARITHMETIC LOGIC
EUL REGISTER-COUNTER,
BITS 1-4

Sheet ____ of ____

Drawing No. AUO-07-212-04



CARD LOCATION: C-22

Issue	Change order	Approved by	Date
2	No.531	P. Kralle	4-18-72

REVISIONS

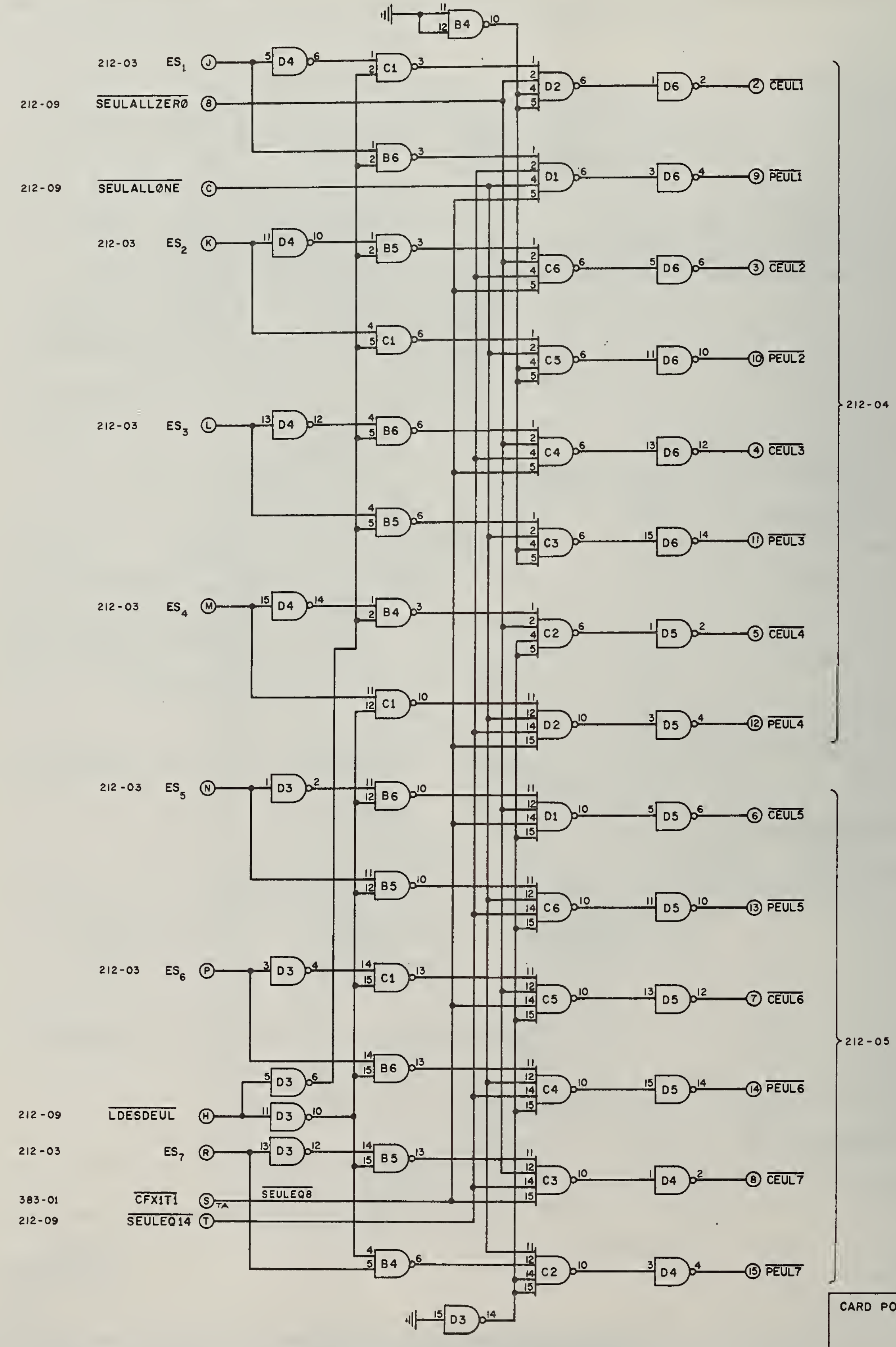
DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For: L.G
Drawn by: T G
Date: 7-6-71
Supersedes dwg.:
Approved by:
Date: 8-6-1971
Reference:

TITLE AU EXPONENT - ARITMETIC LOGIC
EUL REGISTER - COUNTER, BITS 5-7 AND
CONDITION DETECT LOGIC

Sheet ____ of ____

Drawing No. AUO-07-212-05



CARD POSITION: C-24

Issue	Change order	Approved by	Date

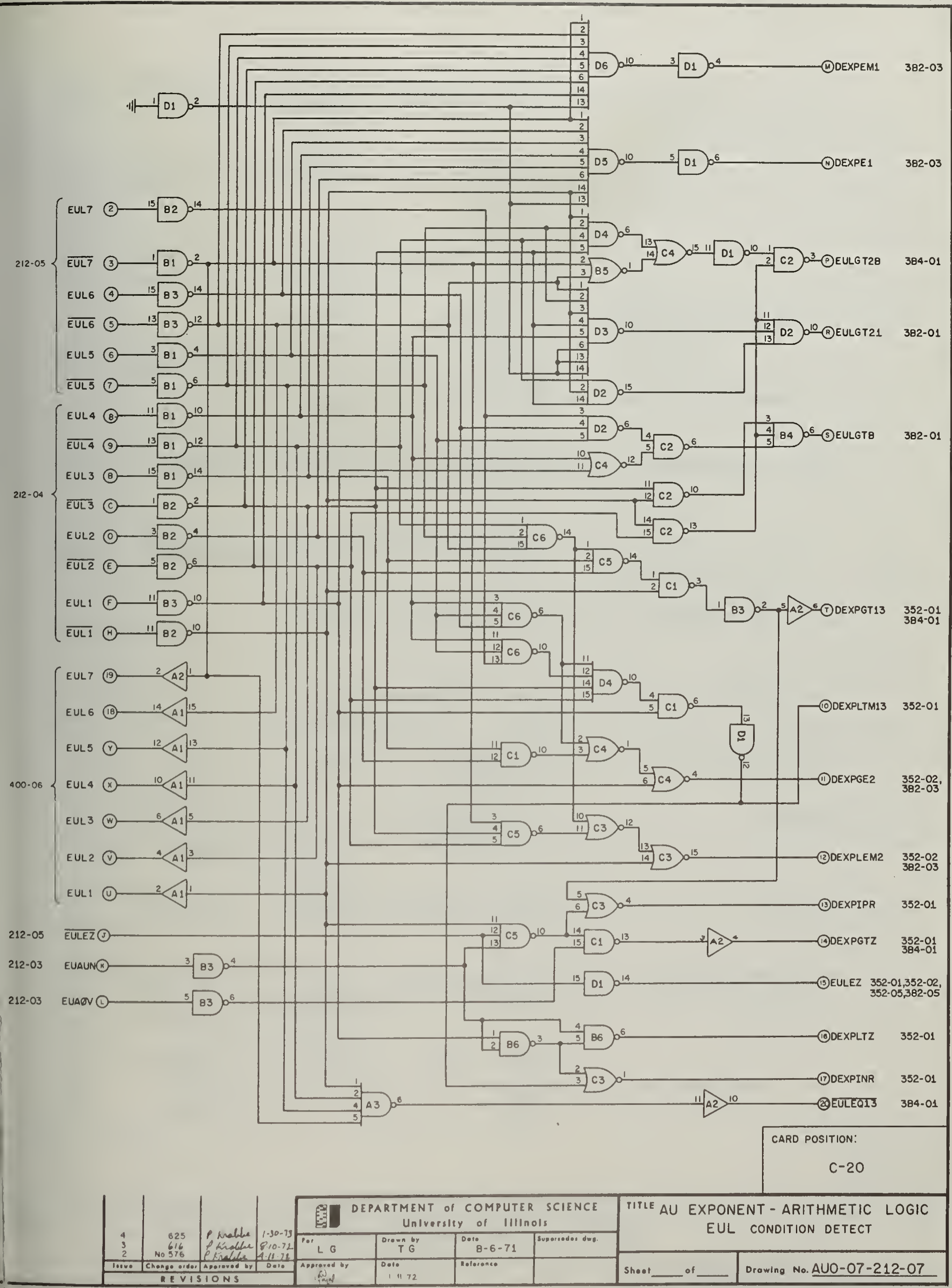
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

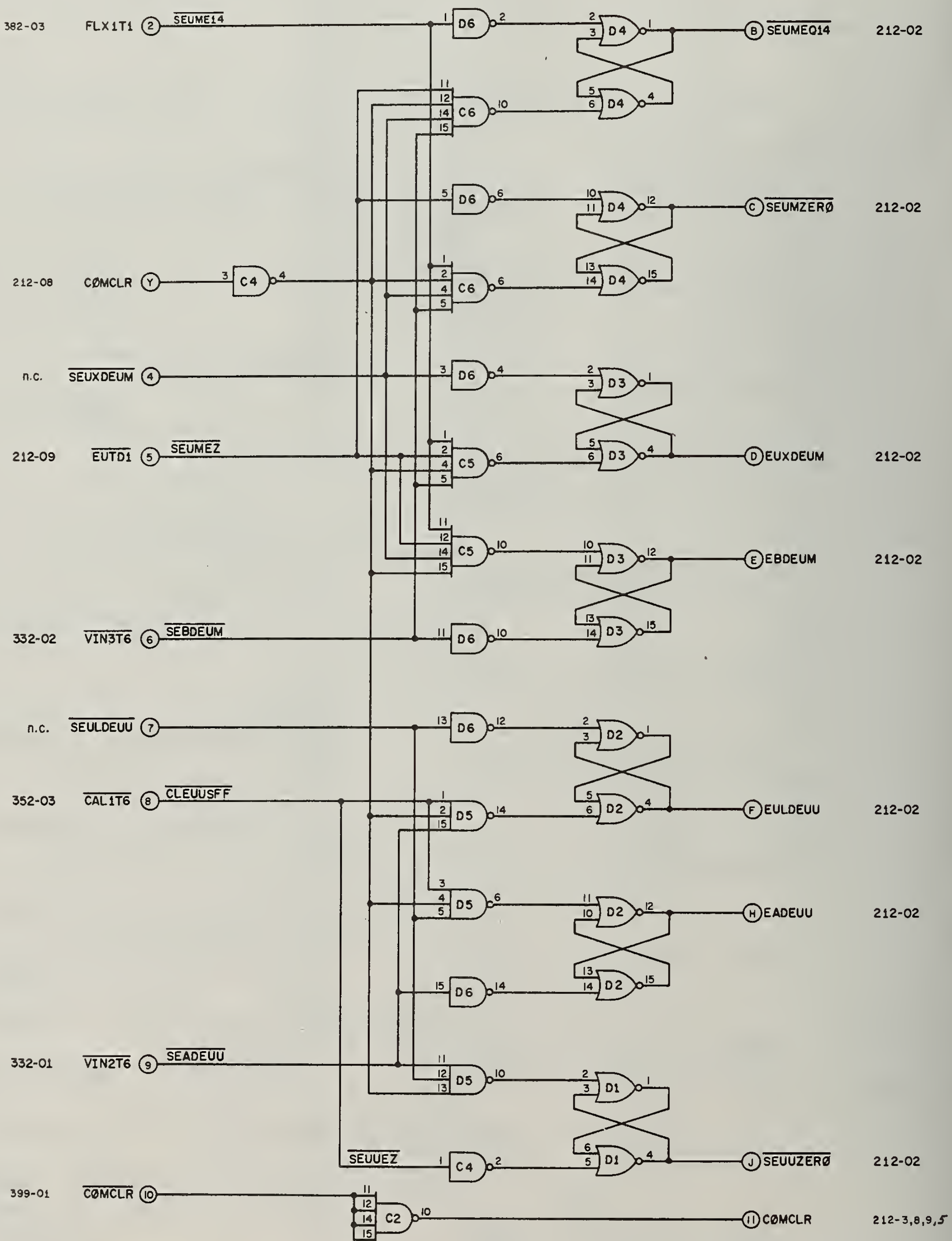
For	Drawn by	Date	Supersedes dwg.
LNG	LRS	5-21-71	
Approved by	Date	Reference	
	6-5-71		

TITLE AU EXPONENT - ARITHMETIC LOGIC
EUL LOAD AND PRESET LOGIC FOR EUL

Sheet ____ of ____ Drawing No. AUO-07-212-06



212-05
EUL7 (2)
$\overline{EUL7}$ (3)
EUL6 (4)
$\overline{EUL6}$ (5)
EUL5 (6)
$\overline{EUL5}$ (7)
212-04
EUL4 (8)
$\overline{EUL4}$ (9)
EUL3 (B)
$\overline{EUL3}$ (C)
EUL2 (D)
$\overline{EUL2}$ (E)
EUL1 (F)
$\overline{EUL1}$ (H)
400-06
EUL7 (19)
EUL6 (18)
EUL5 (Y)
EUL4 (X)
EUL3 (W)
EUL2 (V)
EUL1 (U)
212-05 $\overline{EULEZ}$ (J)
212-03 EUAUN (K)
212-03 EUAØV (L)
(M) DEXPEM1 382-03
(N) DEXPE1 382-03
(P) EULGT28 384-01
(R) EULGT21 382-01
(S) EULGT8 382-01
(T) DEXPGT13 352-01, 384-01
(10) DEXPLTM13 352-01
(11) DEXPGE2 352-02, 382-03
(12) DEXPLEM2 352-02, 382-03
(13) DEXPIPR 352-01
(14) DEXPGTZ 352-01, 384-01
(15) EULEZ 352-01, 352-02, 352-05, 382-05
(16) DEXPLTZ 352-01
(17) DEXPINR 352-01
(20) $\overline{EULEQ13}$ 384-01
CARD POSITION:
C-20
DEPARTMENT of COMPUTER SCIENCE
University of Illinois
TITLE AU EXPONENT - ARITHMETIC LOGIC
EUL CONDITION DETECT
For L G
Drawn by T G
Date 8-6-71
Supersedes dwg.
Approved by
Date 1 11 72
Reference
Sheet ___ of ___
Drawing No. AU0-07-212-07
4 625 P Krabbe 1-30-73
3 616 P Krabbe 8-10-72
2 No 576 P Krabbe 4-11-72
Issue | Change order | Approved by | Date
REVISIONS



CARD POSITION: C28

2	NO. 590	P. Knoblk	5-25-72
Issue	Change order	Approved by	Date

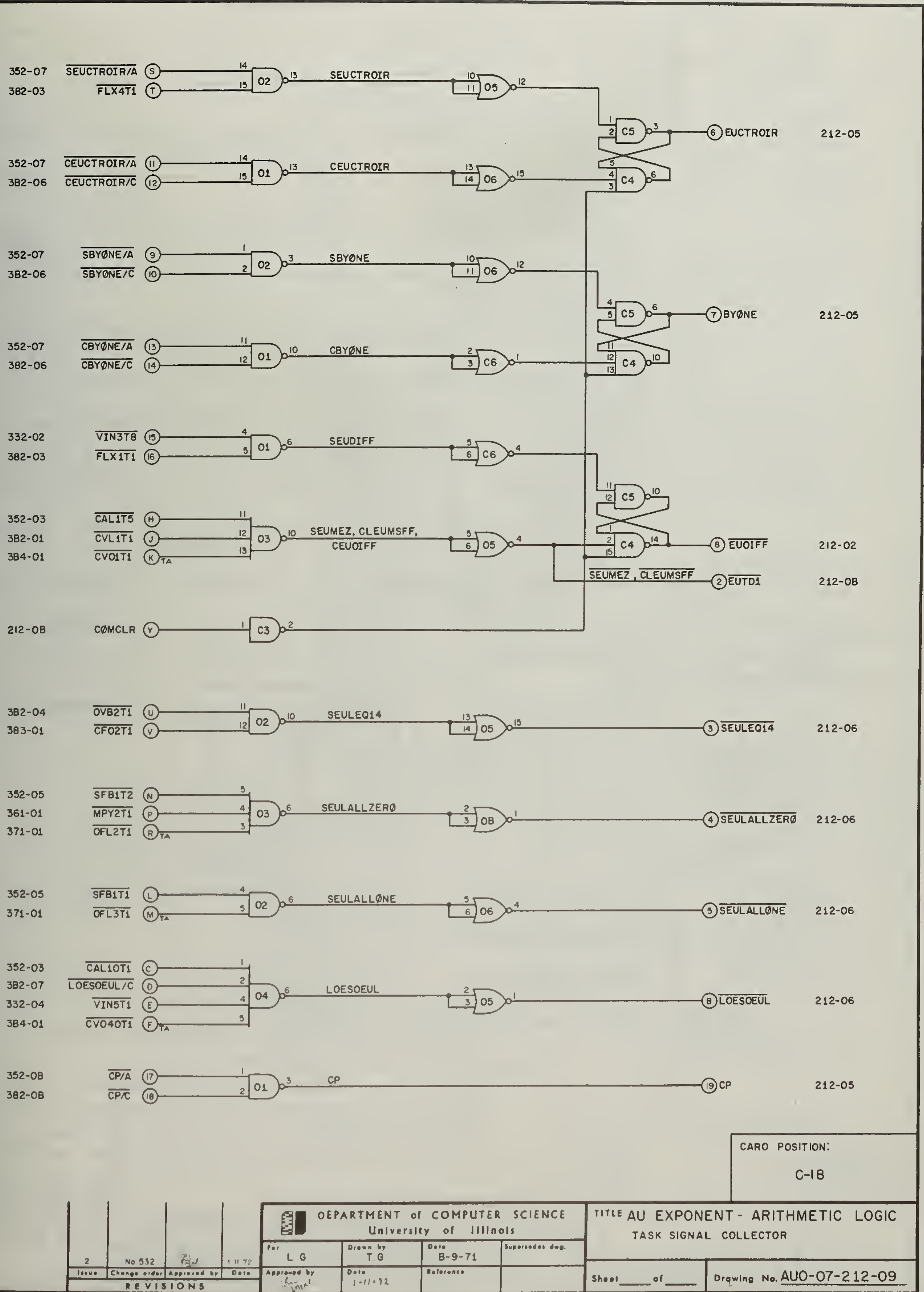
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

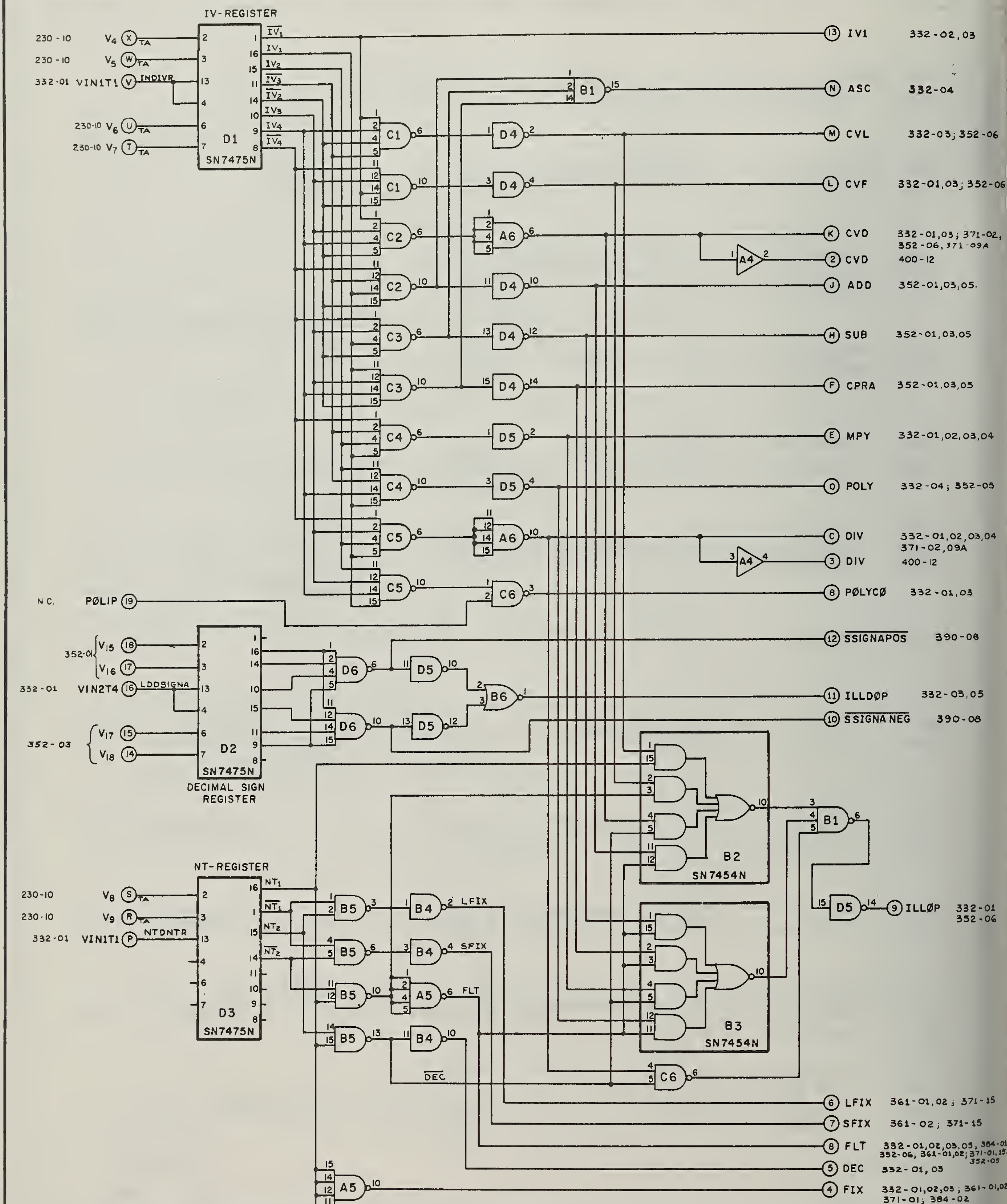
For	Drawn by	Date	Supersedes dwg.
L.G.	T. G.	8-12-71	
Approved by	Date	Reference	
[signature]	1-11-72		

TITLE AU EXPONENT-ARITHMETIC LOGIC
EUM AND EUU SELECTOR SWITCHES

Sheet ____ of ____ Drawing No. AUO-07-212-08



352-07 SEUCTROIR/A (S)
382-03 FLX4T1 (T)
SEUCTROIR
(6) EUCTROIR 212-05
352-07 CEUCTROIR/A (11)
382-06 CEUCTROIR/C (12)
CEUCTROIR
352-07 SBYØNE/A (9)
382-06 SBYØNE/C (10)
SBYØNE
(7) BYØNE 212-05
352-07 CBYØNE/A (13)
382-06 CBYØNE/C (14)
CBYØNE
332-02 VIN3T8 (15)
382-03 FLX1T1 (16)
SEUDIFF
352-03 CAL1T5 (H)
382-01 CVL1T1 (J)
384-01 CVO1T1 (K) TA
SEUMEZ, CLEUMSFF, CEUOIFF
(8) EUOIFF 212-02
SEUMEZ, CLEUMSFF
(2) EUTD1 212-08
212-08 CØMCLR (Y)
382-04 OVB2T1 (U)
383-01 CFO2T1 (V)
SEULEQ14
(3) SEULEQ14 212-06
352-05 SFB1T2 (N)
361-01 MPY2T1 (P)
371-01 OFL2T1 (R) TA
SEULALLZERØ
(4) SEULALLZERØ 212-06
352-05 SFB1T1 (L)
371-01 OFL3T1 (M) TA
SEULALLØNE
(5) SEULALLØNE 212-06
352-03 CAL10T1 (C)
382-07 LOESOEUL/C (D)
332-04 VIN5T1 (E)
384-01 CVO40T1 (F) TA
LOESOEUL
(8) LOESOEUL 212-06
352-08 CP/A (17)
382-08 CP/C (18)
CP
(19) CP 212-05
CARD POSITION:
C-18
DEPARTMENT of COMPUTER SCIENCE
University of Illinois
TITLE AU EXPONENT - ARITHMETIC LOGIC
TASK SIGNAL COLLECTOR
For L G
Drawn by T.G
Date 8-9-71
Supersedes dwg.
Approved by
Date 1-11-72
Reference
Sheet ___ of ___
Drawing No. AUO-07-212-09
2 No 532 1 11 72
Issue Change order Approved by Date
REVISIONS



CARD LOCATION: C-48

Issue	Change order	Approved by	Date
2	No 577	P Kralike	4-12-72

REVISIONS

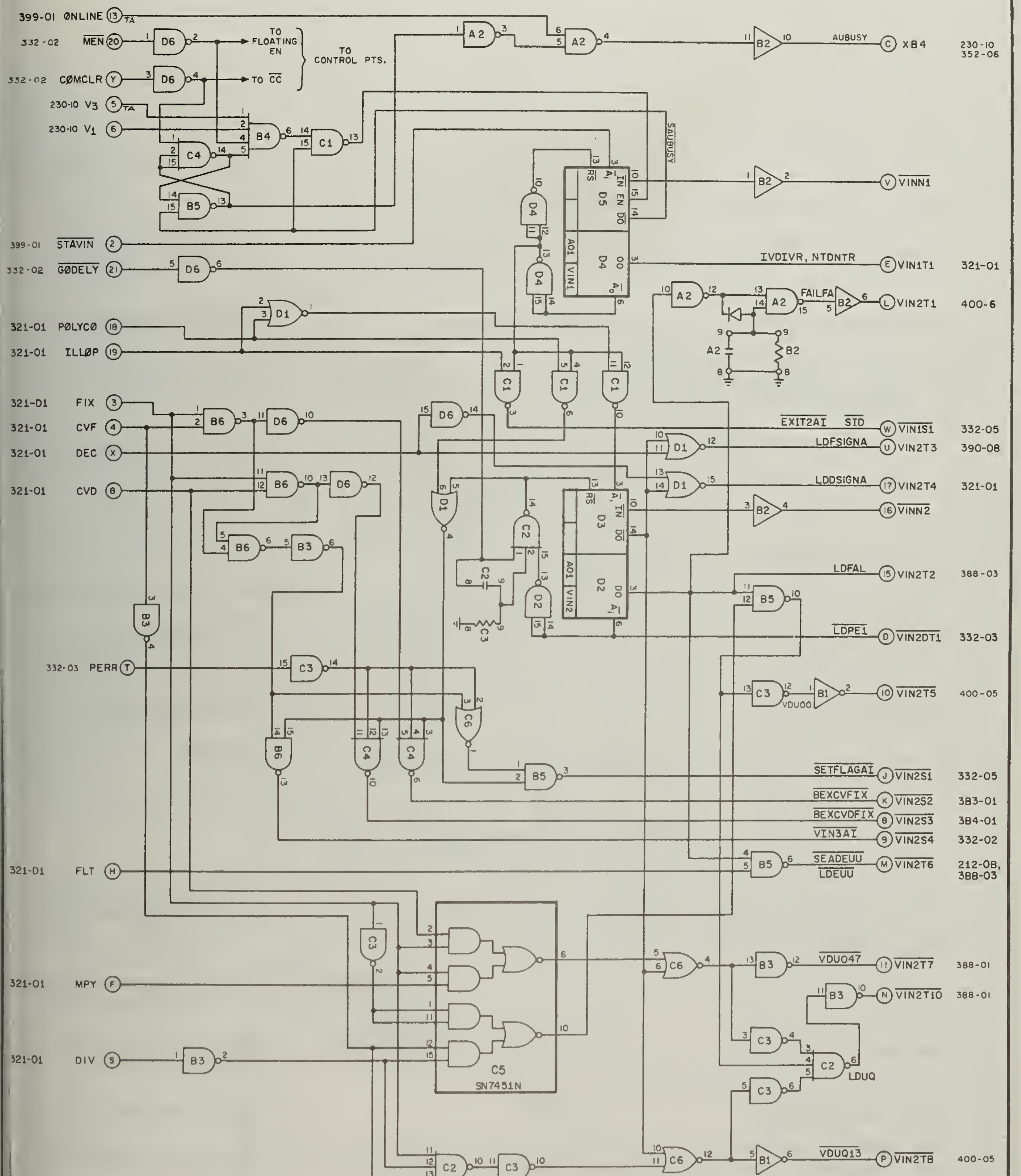
DEPARTMENT of COMPUTER SCIENCE
University of Illinois

TITLE AU DECODE CONTROL LOGIC
OP CODE AND NUMBER TYPE DECODER

For	Drawn by	Date	Supersedes dwg.
L.G	S.Z	10-7-71	
Approved by	**Date**	**Reference**	
[illegible]	1-41-72		

Sheet ____ of ____

Drawing No. AU0-07-321-01



CARD LOCATION:

C-56

Issue	Change order	Approved by	Date
3	No 626	P Krable	1-30-73
2	No 543	P Krable	4-4-72

REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

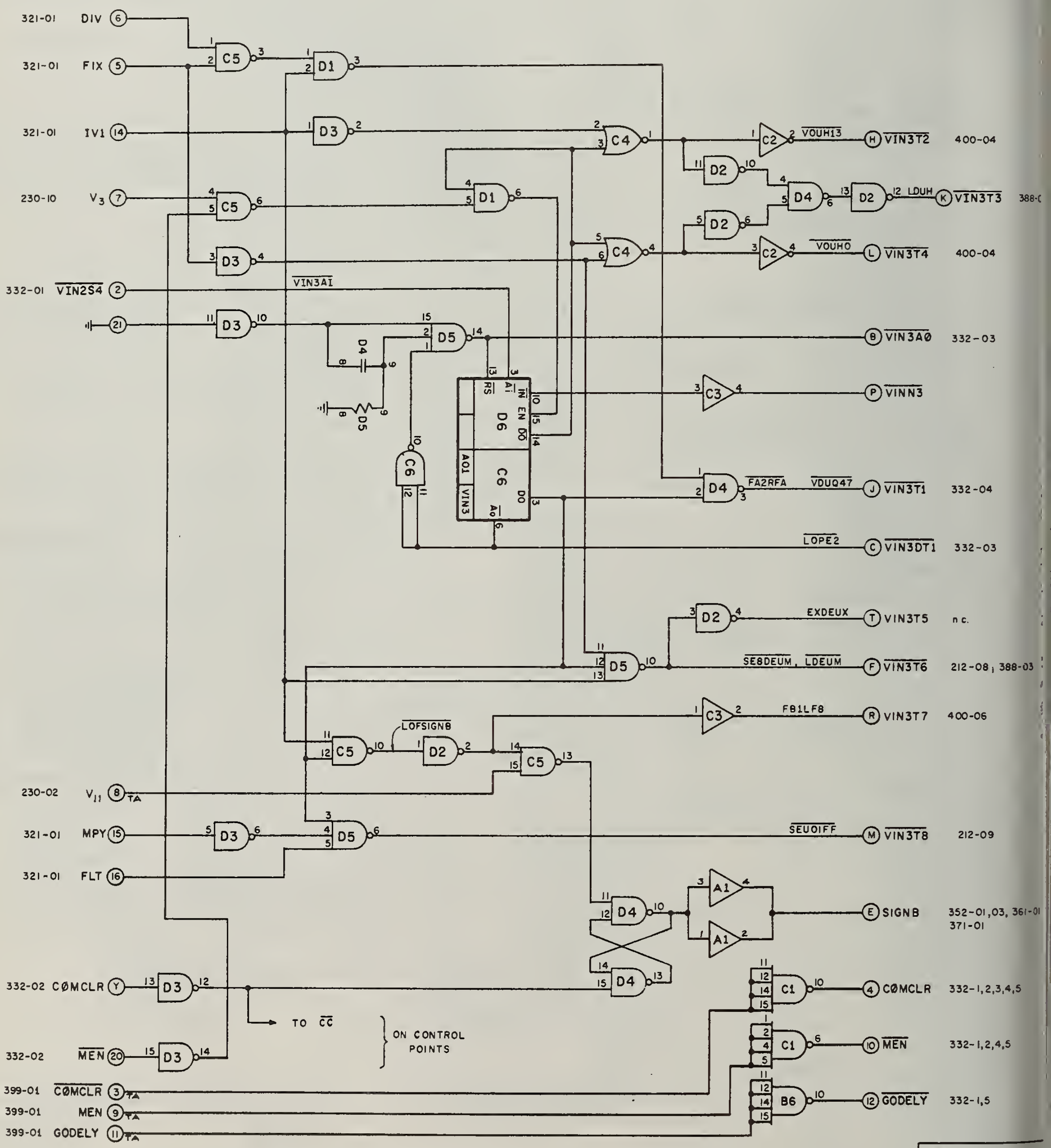
For: L. GOYAL | Drawn by: T. G | Date: 9-8-71 | Supersedes dwg.:

Approved by: Goyal | Date: 1-4-73 | Reference:

TITLE AU OPERANDS-LOAD CONTROL LOGIC
FIRST WORD LOAD AND BRANCH
-FIWLOB-

Sheet ____ of ____

Drawing No. AUO-07-332-01



CARD LOCATION: C54

Issue	Change order	Approved by	Date
4	No. 607	P. Krabbe	8-10-72
3	No. 591	P. Krabbe	5-25-72
2	No. 544	P. Krabbe	4-12-72

REVISIONS

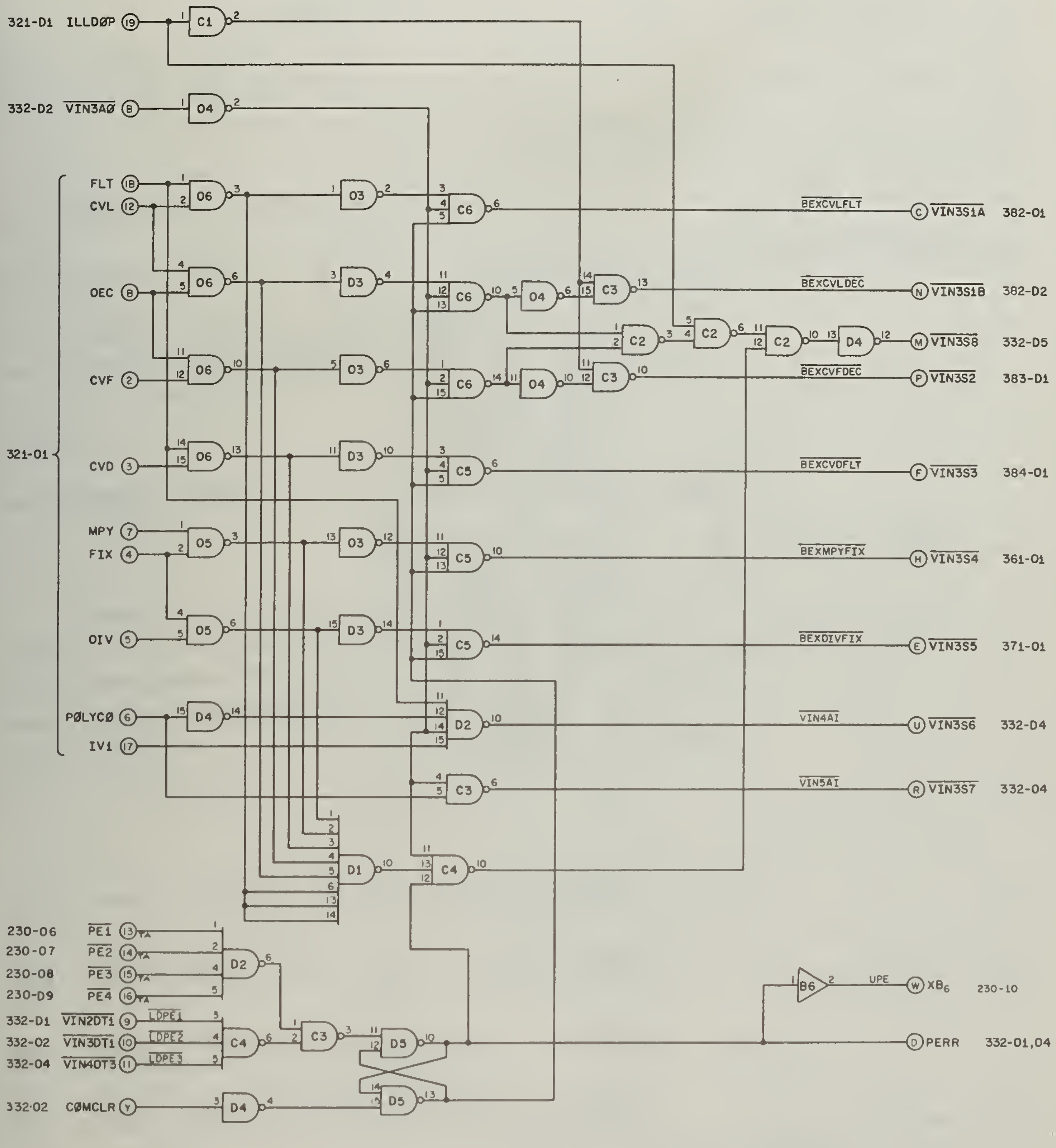
DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L.G	S Z.	10-13-71	
Approved by	Date	Reference	
[illegible]	1-9-72		

TITLE AU OPERANDS-LOAD CONTROL LOGIC
SECOND WORD LOAD
-SEWLO-

Sheet _____ of _____

Drawing No. AU0-07-332-02



CARO LOCATION: C-60

2	NO. 549	P. Kraatz	4-12-72
Issue	Change order	Approved by	Date

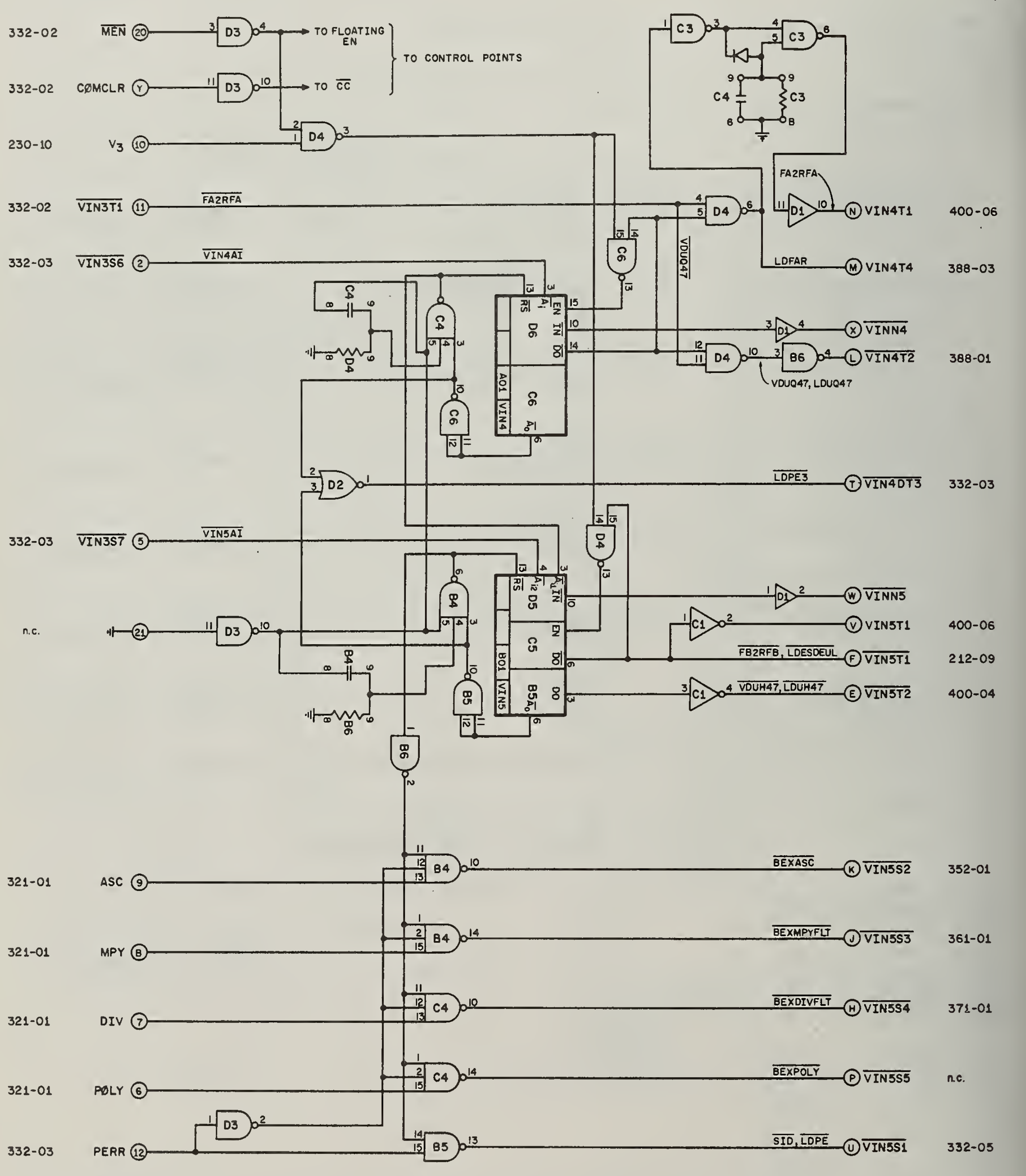
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L. G.	T. G	10-13-71	
Approved by	Date	Reference	
	1-9-72		

TITLE AU OPERANDS-LOAD CONTROL LOGIC
BRANCH TO APPROPRIATE INSTRUCTION
-BRAIN-

Sheet____ of ____ Drawing No. AUO-07-332-03



CARD LOCATION:
C-58

Issue	Change order	Approved by	Date
3	NO. 627	P. Kralle	1-30-73
2	NO. 545	P. Kralle	4-12-72

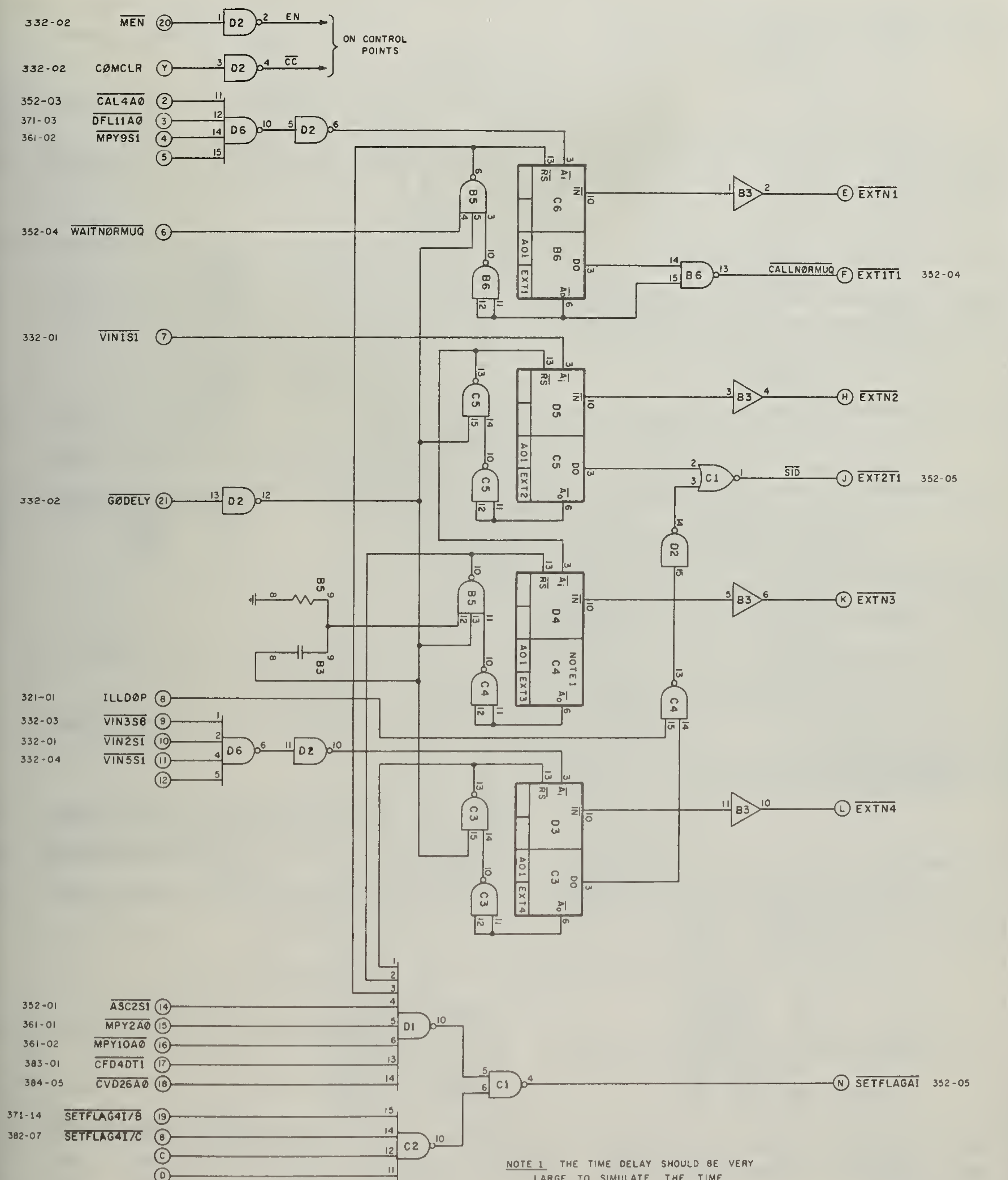
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L.G.	T.G.	10-15-71	
Approved by	Date	Reference	
[signature]	1-11-72		

TITLE AU OPERANDS-LOAD CONTROL LOGIC
THIRD AND FOURTH WORDS LOAD AND BRANCH
-TIFOWLOB-

Sheet ____ of ____ Drawing No. AUO-07-332-04



NOTE 1 THE TIME DELAY SHOULD BE VERY LARGE TO SIMULATE THE TIME NECESSARY TO LOAD ALL FOUR OPERANDS IN WORST CASE

CARD LOCATION: C-52

Issue	Change order	Approved by	Date
2	NO. 550	P Krable	4-12-72

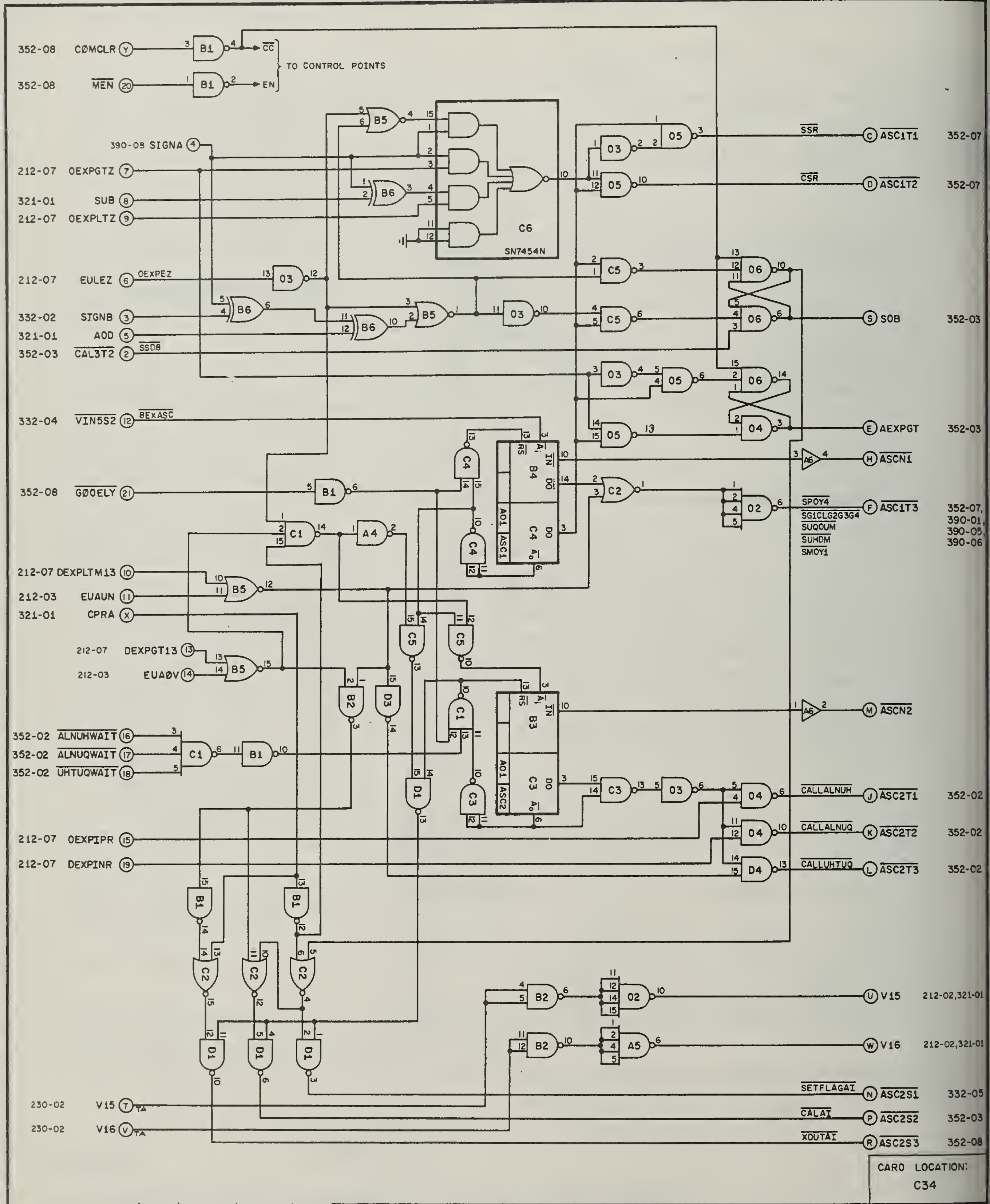
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L G	S Z	10-18-71	
Approved by	Date	Reference	

TITLE AU-REFOTRAN CONTROL LOGIC
EXIT

Sheet ____ of ____ Drawing No. AUO-07-332-05



CARD LOCATION: C34

Issue	Change order	Approved by	Date
3	No 623	P. Krabbe	10-5-72
2	NO. 592	P. Krabbe	5-25-72

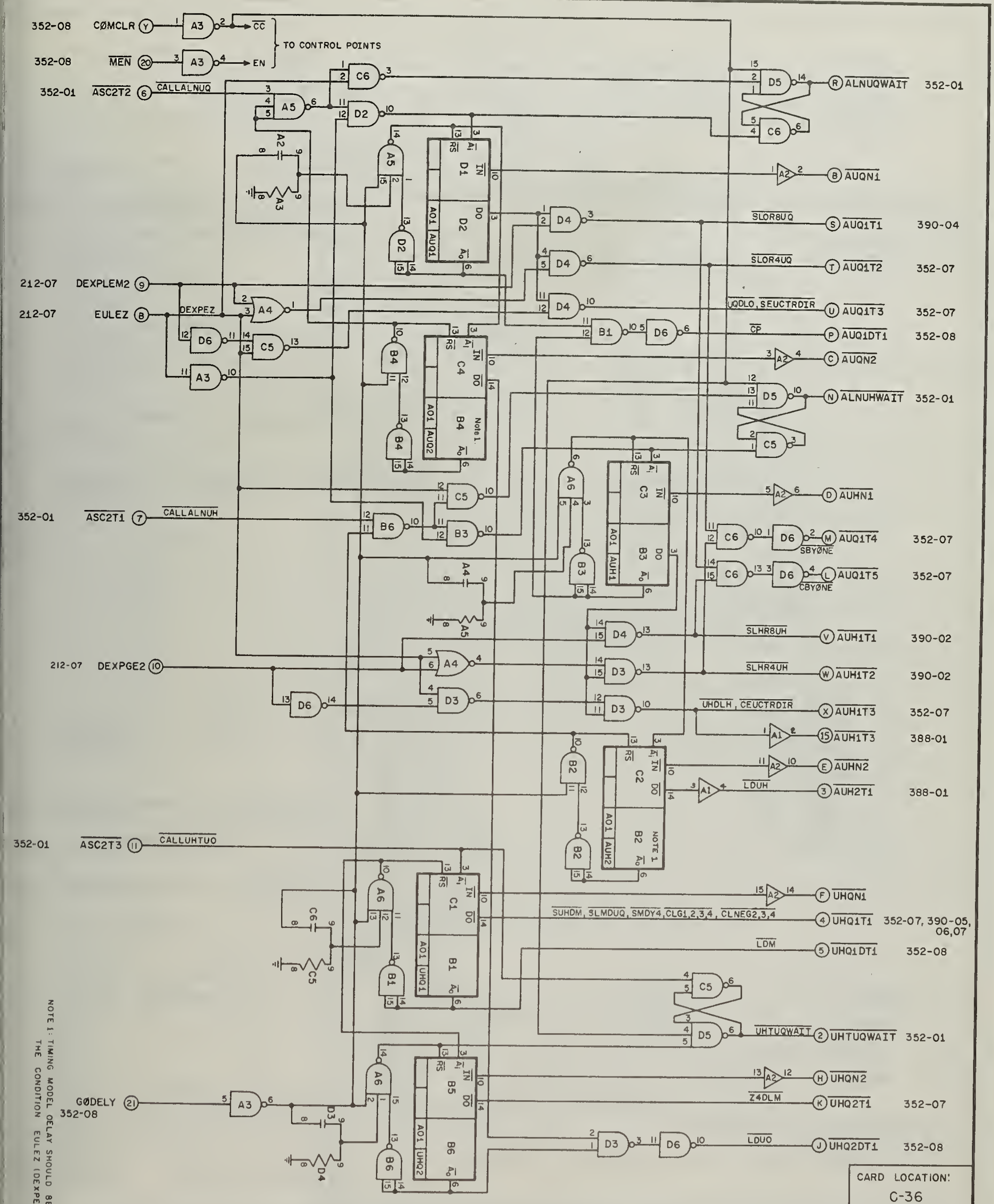
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L. G.	T. G.	10-29-71	
Approved by	Date	Reference	
Gyal	1-11-72		

TITLE AU - ASC CONTROL LOGIC
OPERANDS ALIGNMENT CALL SET UP
- OPACAS -

Sheet ____ of ____ Drawing No. AUO-07-352-01



CARD LOCATION: C-36

Issue	Change order	Approved by	Date
	608	P Krabbe	8-10-72

REVISIONS

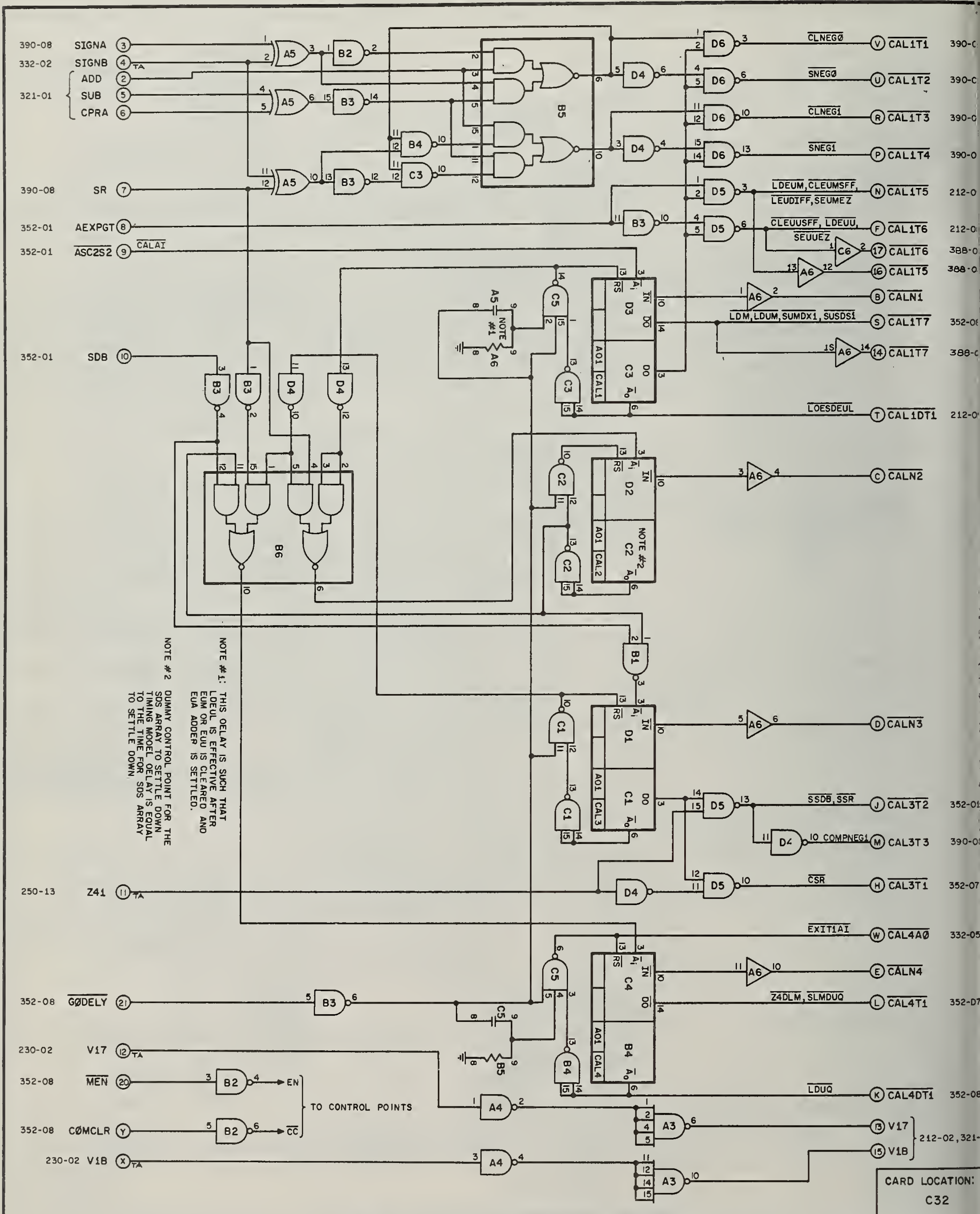
DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L. G.	T. G.	10-20-71	
Approved by	Date	Reference	
[illegible]	1-11-72		

TITLE AU - ASC CONTROL LOGIC
ALIGN UQ, ALIGN UH, UH TO UQ
-ALNUH, ALNUQ, UHTUQ-

Sheet ____ of ____

Drawing No. AUO-07-352-02



Issue	Change order	Approved by	Date
3	No. 609		8-10-72
2	No. 593		5-25-72

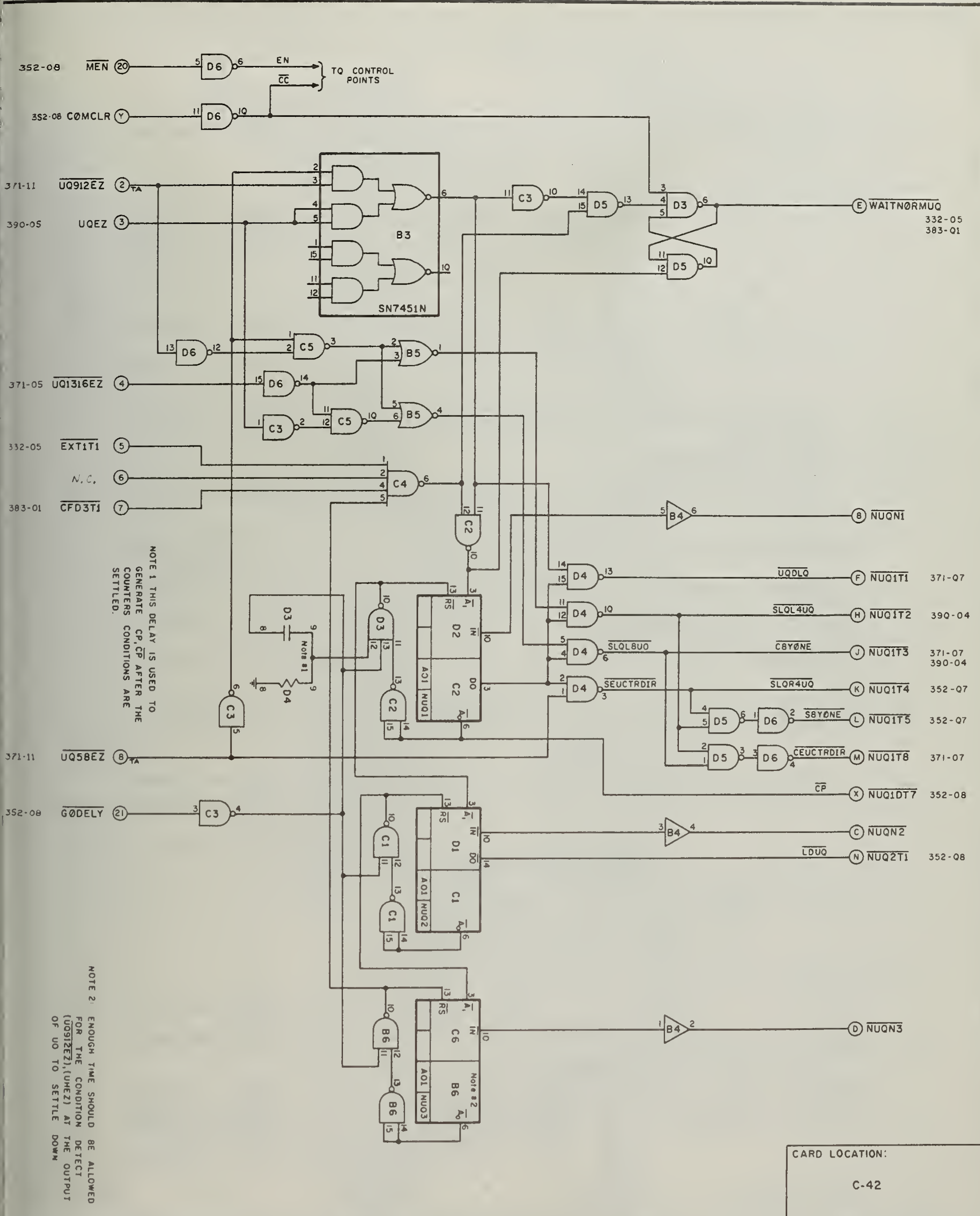
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For: L. G. — Drawn by: T. G. — Date: 11-12-71 — Supersedes dwg.: — Approved by: — Date: 1-11-72 — Reference:

TITLE AU - ASC, CONTROL LOGIC
SUM OR DIFFERENCE CALCULATION
- CAL -

Sheet ____ of ____ Drawing No. AUO-07-352-03



CARD LOCATION:

C-42

Issue	Change order	Approved by	Date
REVISIONS			

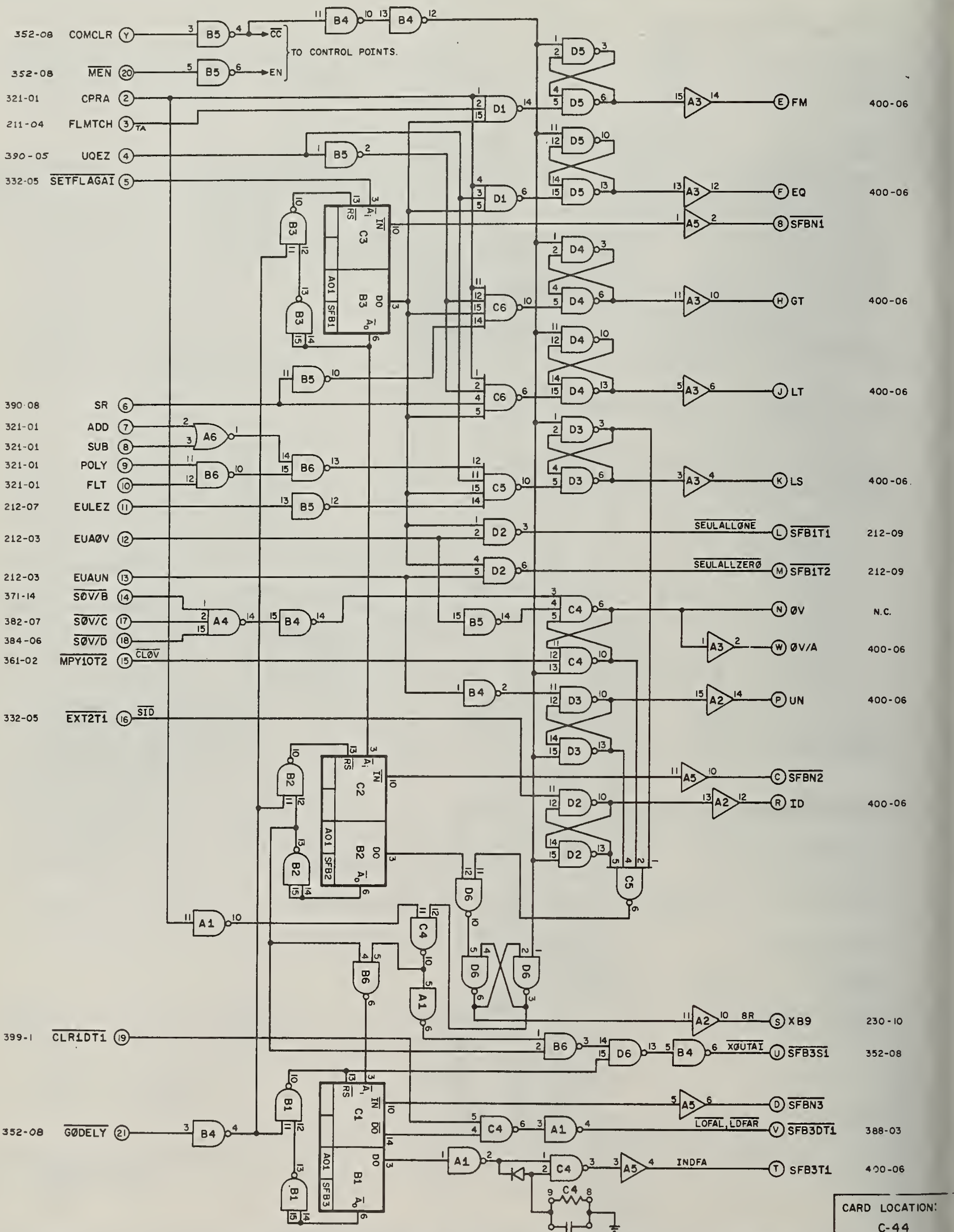
DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L G.	S.Z	11-24-71	
Approved by	Date	Reference	

TITLE AU-REFOTRAN CONTROL LOGIC
NORMALIZE UQ
-NORMUQ-

Sheet ____ of ____

Drawing No. AUO-07-352-04



CARD LOCATION:
C-44

Issue	Change order	Approved by	Date
3	No. 828	P. Krable	1-30-73
2	No. 547	P. Krable	4-12-72

REVISIONS

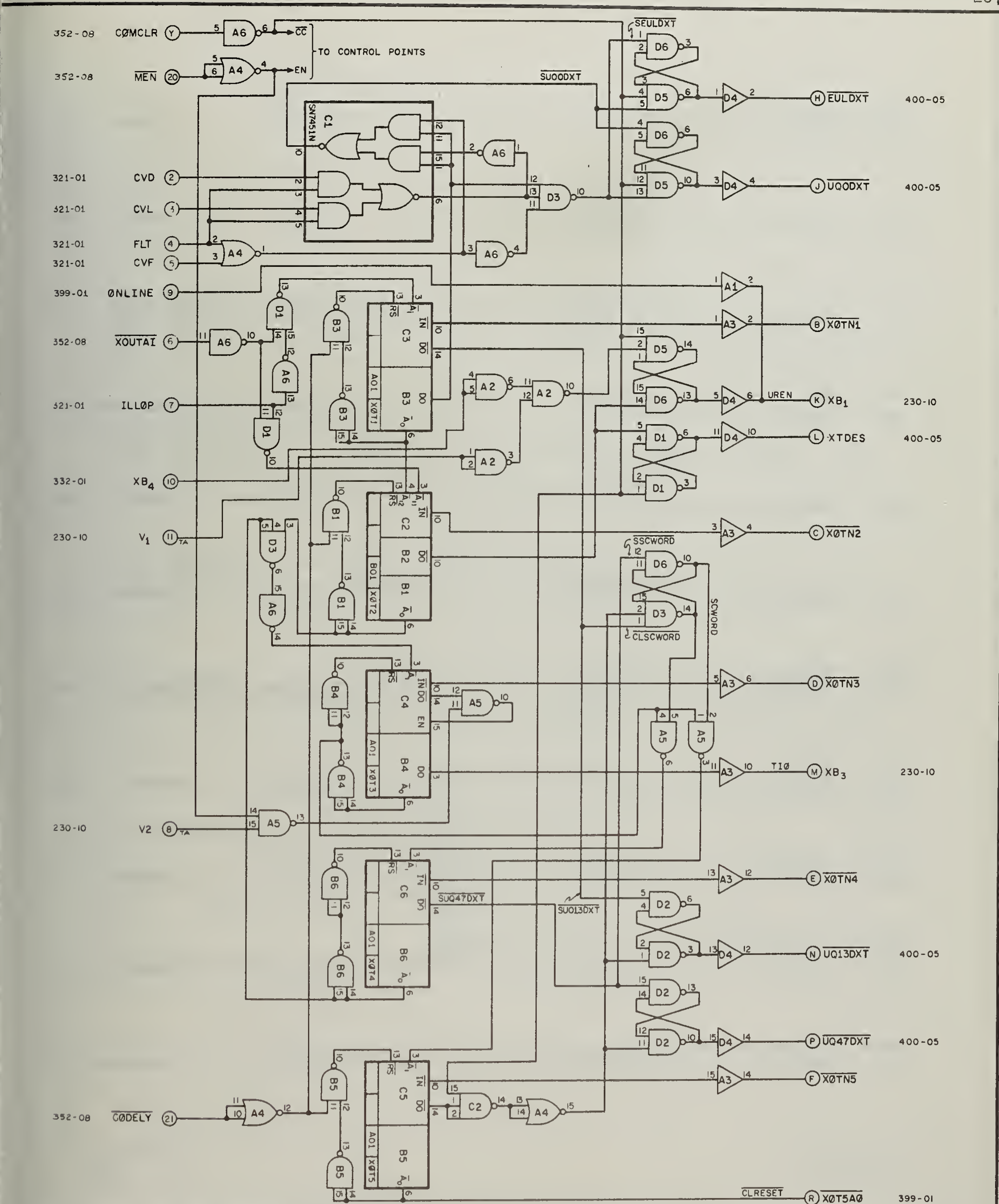
DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For: L. G. | Drawn by: T. G. | Date: 1-12-72 | Supersedes dwg.

Approved by: | Date: 18 1 1972 | Reference

TITLE AU-REFØTRAN CONTROL LOGIC
SET FLAGS, BOGUS AND STATUS INDICATORS
SFBIN

Sheet ____ of ____ Drawing No. AU0-07-352-05



CARD LOCATION:
C-46

Issue	Change order	Approved by	Date
3	No 653	P Kraffe	1-30-73
2	No 548	P Kraffe	4-12-72

REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

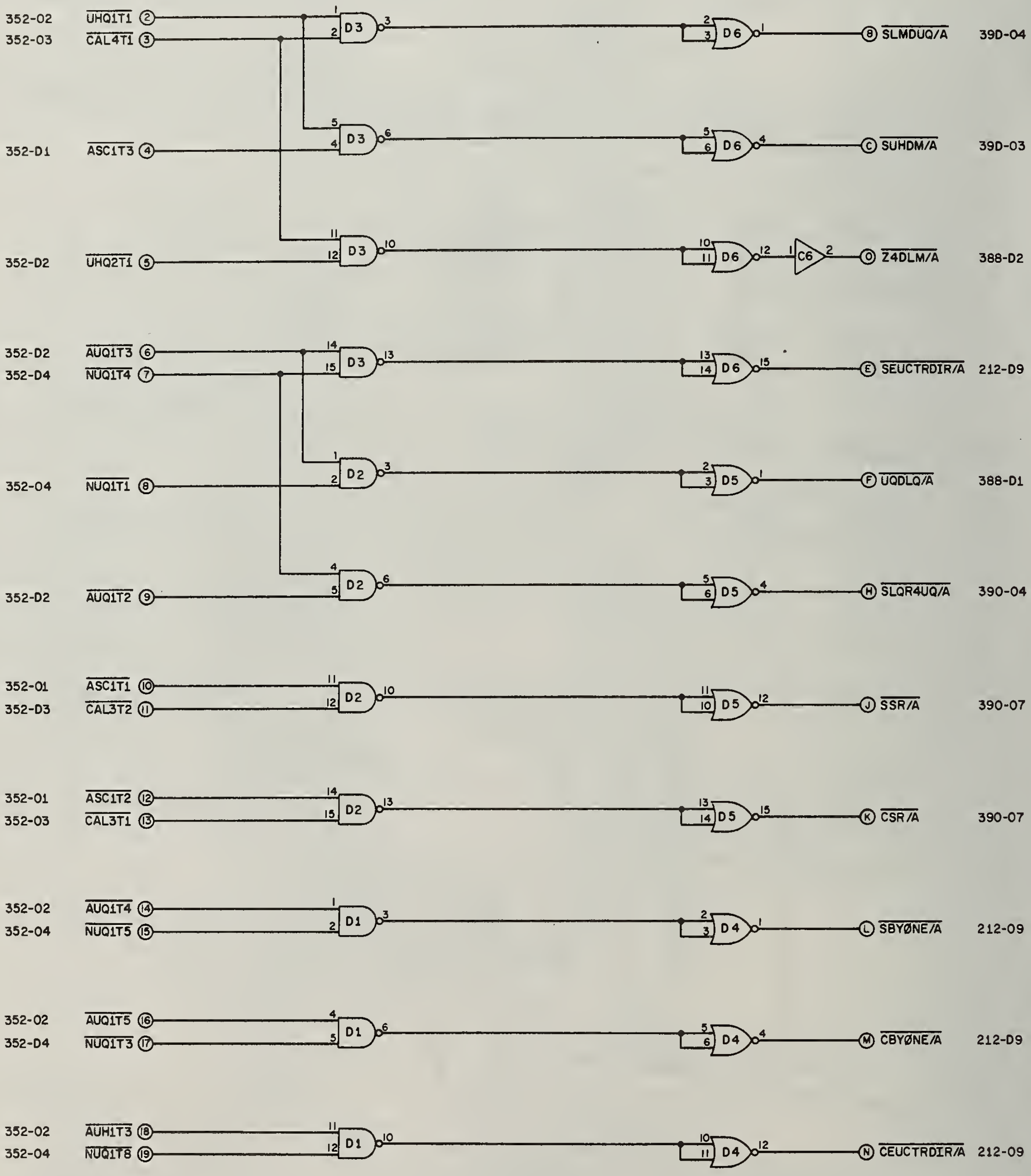
For	Drawn by	Date	Superseded dwg.
L G	T G	1-17-72	

Approved by	Date	Reference
	1-27-1972	

TITLE AU-REFØTRAN CONTROL LOGIC
TRANSFER RESULTS TO PROCESSOR
XØUT

Sheet ____ of ____

Drawing No. AUO-07-352-06



CARD LOCATION:

C-38

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

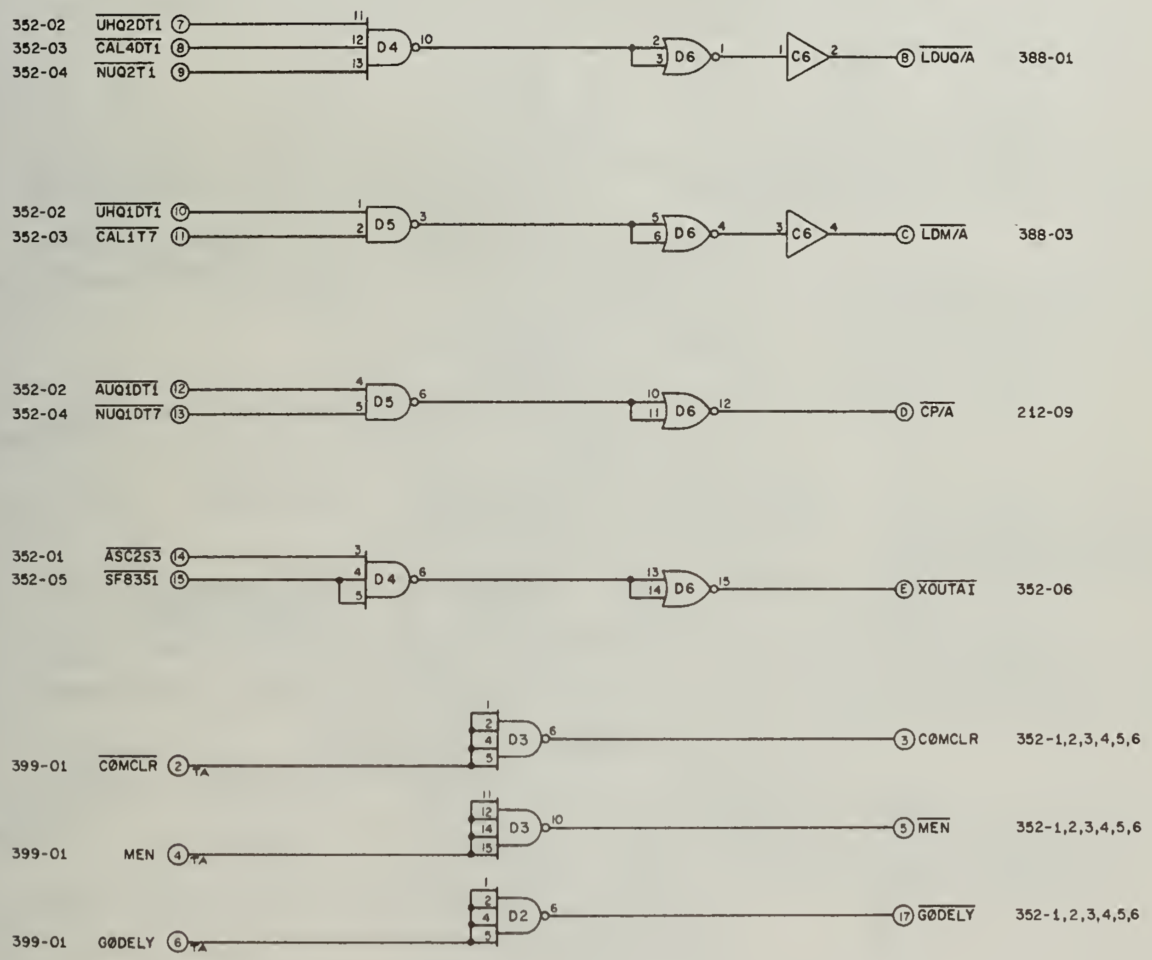
Issue	Change order	Approved by	Date
2	No. 617	P. Kuehle	8-10-72

REVISIONS

For	Drawn by	Date	Supersedes dwg.
L.G.	T. G.	7-29-71	
Approved by	**Date**	**Reference**	
Gray	1-11-72		

TITLE AU-ASC CONTROL LOGIC
TASK SIGNAL COLLECTOR "A1"

Sheet ____ of ____ Drawing No. AUO-07-352-07



CARD LOCATION:

C40

Issue	Change order	Approved by	Date
3	No. 618	P. Kraska	8-10-72
2	No. 594	P. Kraska	5-25-72

REVISIONS

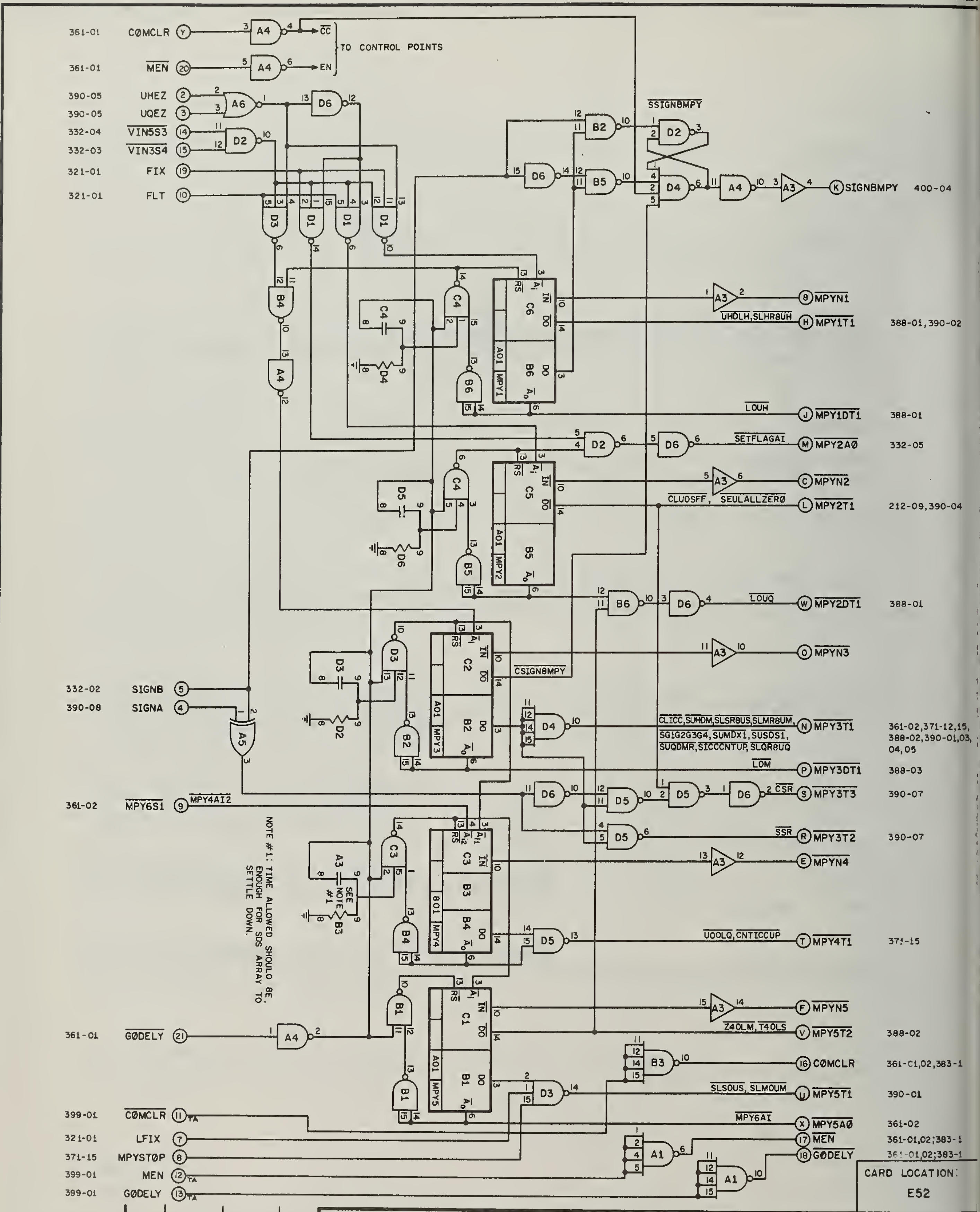
DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L. G.	T. G.	8-3-71	
Approved by	Date	Reference	
Rayal	1-11-72		

TITLE AU-ASC CONTROL LOGIC
TASK SIGNAL COLLECTOR "A2"

Sheet ____ of ____

Drawing No. AUO-07-352-08



DEPARTMENT of COMPUTER SCIENCE
University of Illinois

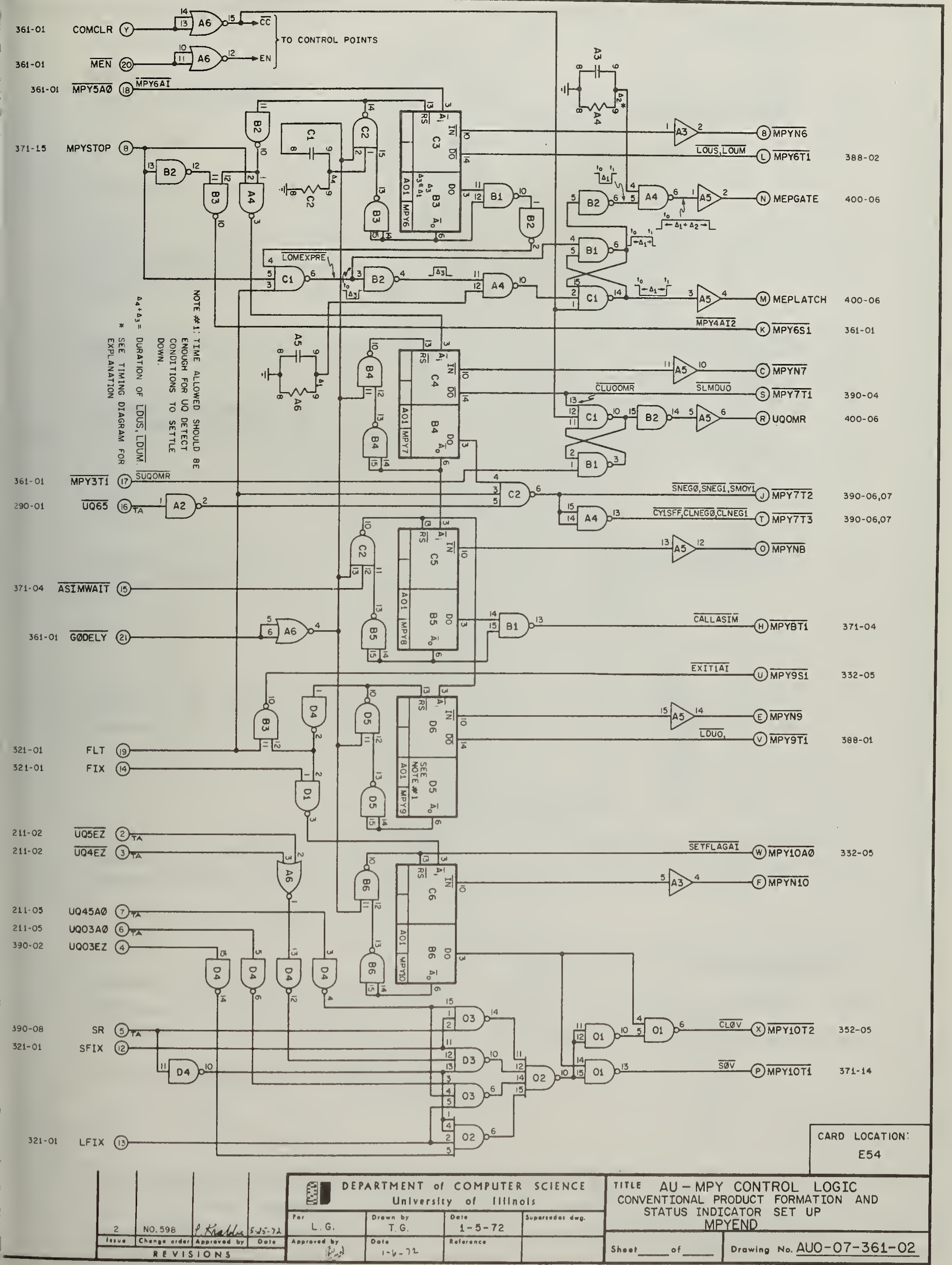
For	Drawn by	Date	Supersedes dwg.
L. G.	T. G.	1-11-72	
Approved by	Date	Reference	
	1 18 1972		

TITLE AU-MPY CONTROL LOGIC
REDUNDANT FORM PRODUCT GENERATION
REPROG

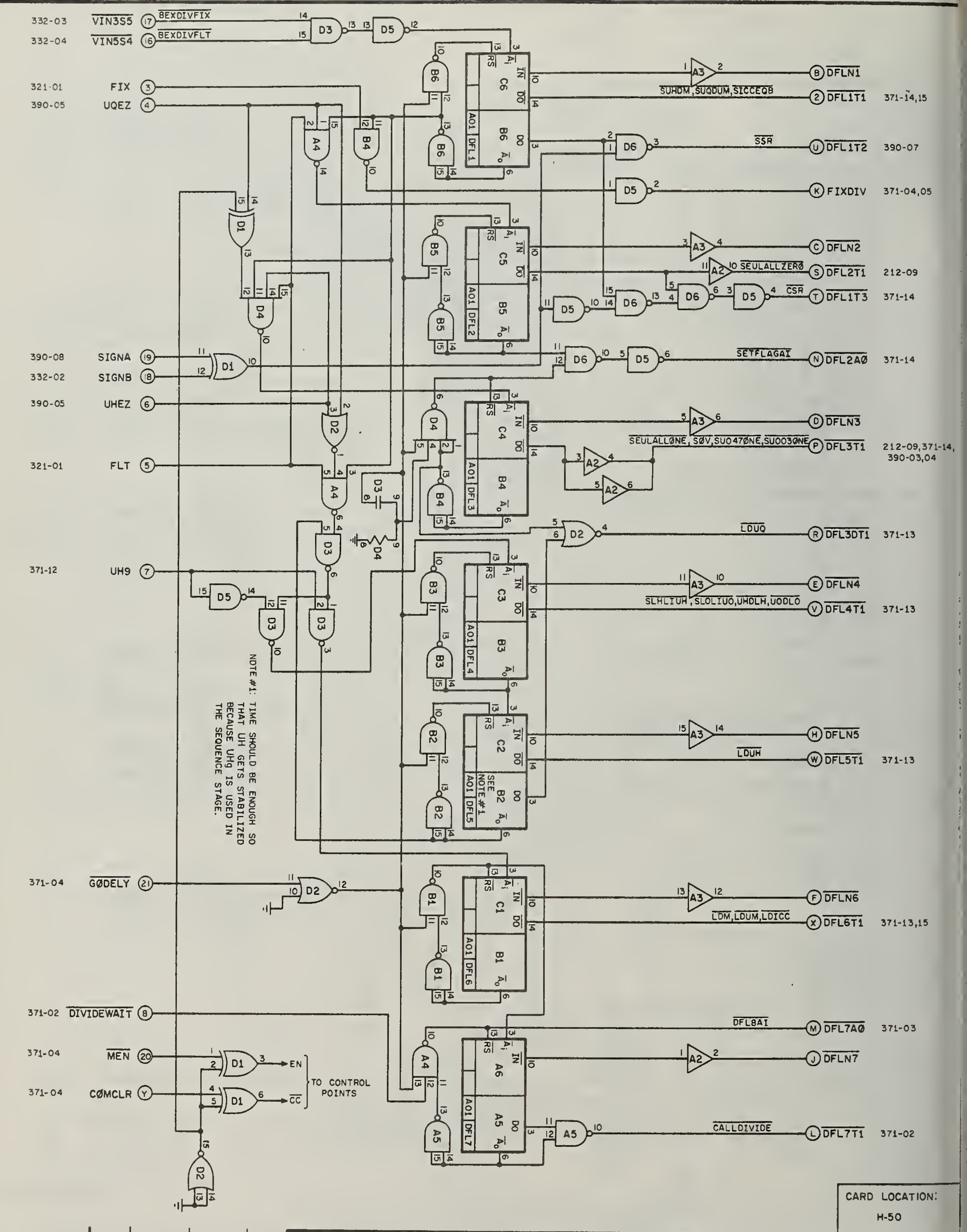
Sheet ____ of ____ Drawing No. AUO-07-361-01

Issue	Change order	Approved by	Date
2	NO. 595	P. Kr[illegible]	5-25-72

REVISIONS



361-01 COMCLR (Y)
361-01 MEN (20)
361-01 MPY5AØ (18) MPY6AI
371-15 MPYSTOP (8)
TO CONTROL POINTS
CC
EN
LOMEXPRE
NOTE #1: TIME ALLOWED SHOULD BE ENOUGH FOR UQ DETECT CONDITIONS TO SETTLE DOWN.
Δ4+Δ3= DURATION OF LDUS, LDUM.
* SEE TIMING DIAGRAM FOR EXPLANATION
361-01 MPY3T1 (17) SUQOMR
290-01 UQ65 (16)
371-04 ASIMWAIT (15)
361-01 GØDELY (21)
321-01 FLT (19)
321-01 FIX (14)
211-02 UQ5EZ (2)
211-02 UQ4EZ (3)
211-05 UQ45AØ (7)
211-05 UQ03AØ (6)
390-02 UQ03EZ (4)
390-08 SR (5)
321-01 SFIX (12)
321-01 LFIX (13)
(8) MPYN6
LOUS, LOUM (L) MPY6T1 388-02
(N) MEPGATE 400-06
(M) MEPLATCH 400-06
MPY4AI2 (K) MPY6S1 361-01
(C) MPYN7
SLMDUO (S) MPY7T1 390-04
CLUOOMR
(R) UQOMR 400-06
SNEGØ, SNEG1, SMOY1 (J) MPY7T2 390-06,07
CY1SFF, CLNEGØ, CLNEG1 (T) MPY7T3 390-06,07
(O) MPYN8
CALLASIM (H) MPY8T1 371-04
EXIT1AI (U) MPY9S1 332-05
(E) MPYN9
LDUO, (V) MPY9T1 388-01
SETFLAGAI (W) MPY10AØ 332-05
(F) MPYN10
CLØV (X) MPY10T2 352-05
SØV (P) MPY10T1 371-14
SEE NOTE #1
CARD LOCATION: E54
DEPARTMENT of COMPUTER SCIENCE
University of Illinois
For L. G.
Drawn by T. G.
Date 1-5-72
Superseded dwg.
Approved by
Date 1-6-72
Reference
TITLE AU – MPY CONTROL LOGIC
CONVENTIONAL PRODUCT FORMATION AND STATUS INDICATOR SET UP
MPYEND
Sheet ____ of ____
Drawing No. AUO-07-361-02
2 NO. 598 5-25-72
Issue Change order Approved by Date
REVISIONS



Issue	Change order	Approved by	Date
3	No 614		8-10-72
2	No. 533		1 11 72

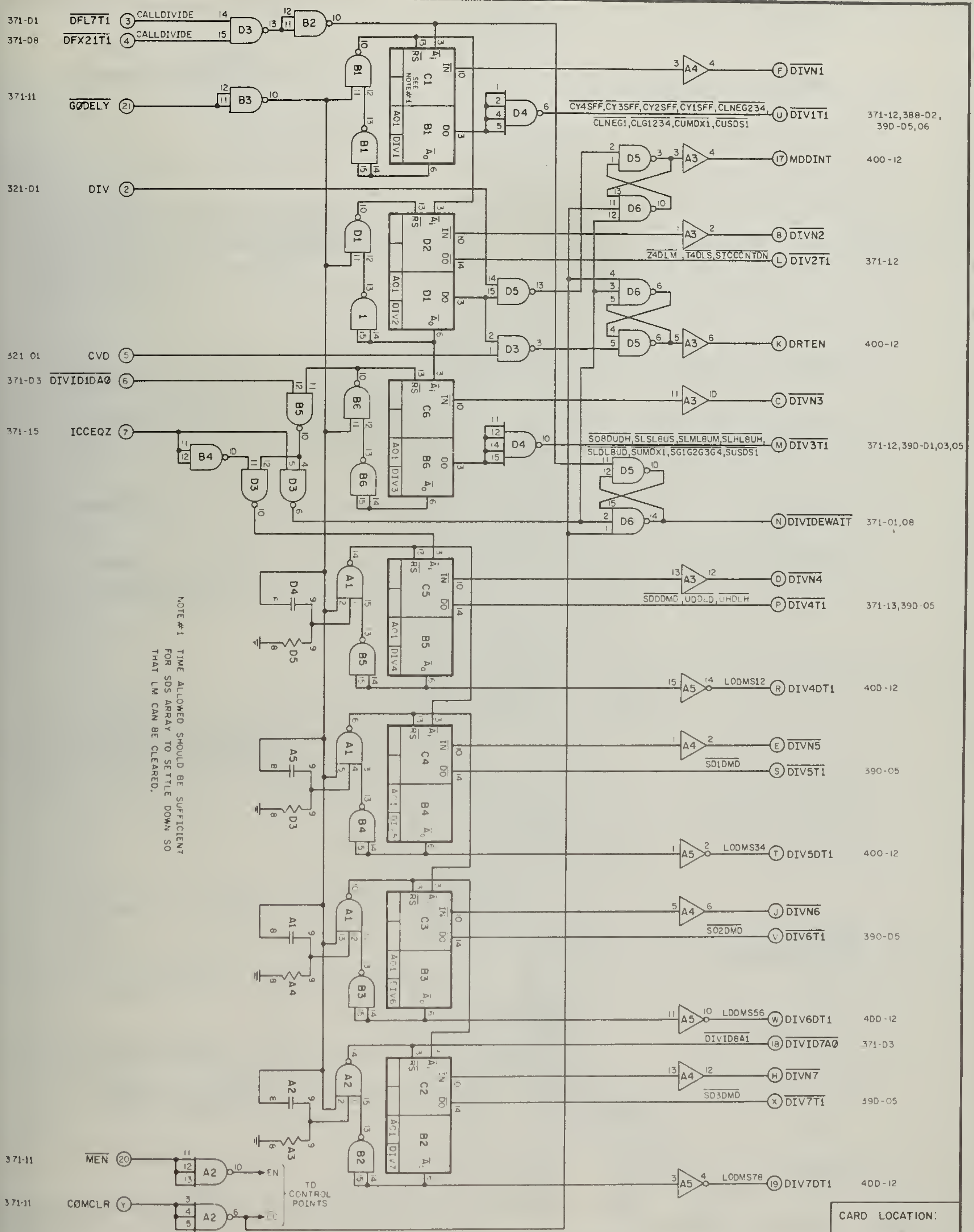
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L. G.	T. G.	1-4-72	
Approved by	Date	Reference	
	1-6-1972		

TITLE AU DIVISION (Fl. Pt.) CONTROL LOGIC
DIVISOR/DIVIDEND ZERO TEST AND SCALING
- DIZETSCAL -

Sheet ____ of ____ Drawing No. AUO-07-371-01



CARD LOCATION: H-38

Issue	Change order	Approved by	Date
2	No. 551	P. Krahle	3-30-72

REVISIONS

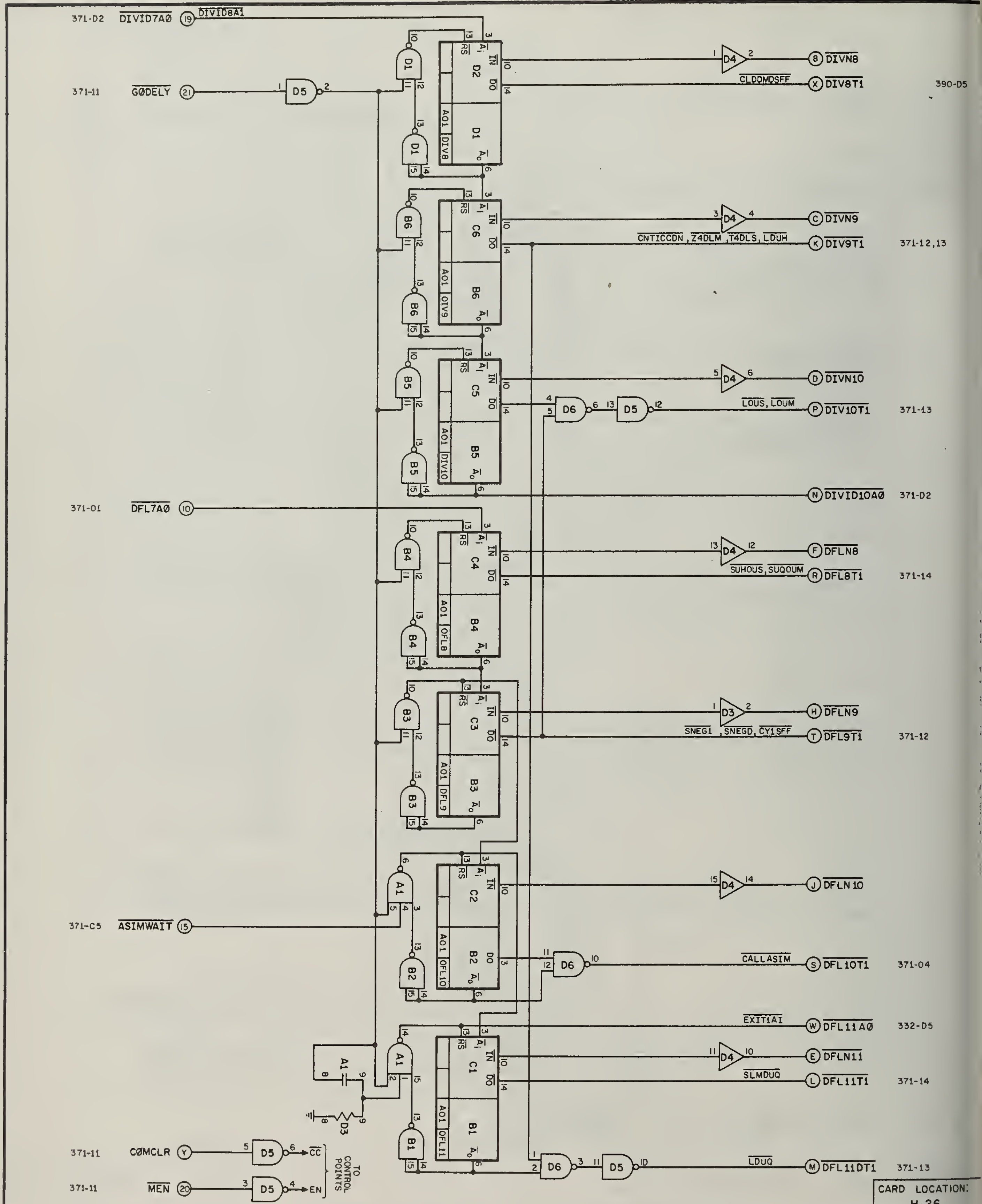
DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg
L G	T G	12-30-71	

Approved by / Date / Reference

TITLE AU DIVISION CONTROL LOGIC
REDUNDANT QUOTIENT GENERATION
- DIVIDE -

Sheet ____ of ____ Drawing No. AUO-07-371-02



CARD LOCATION: H-26

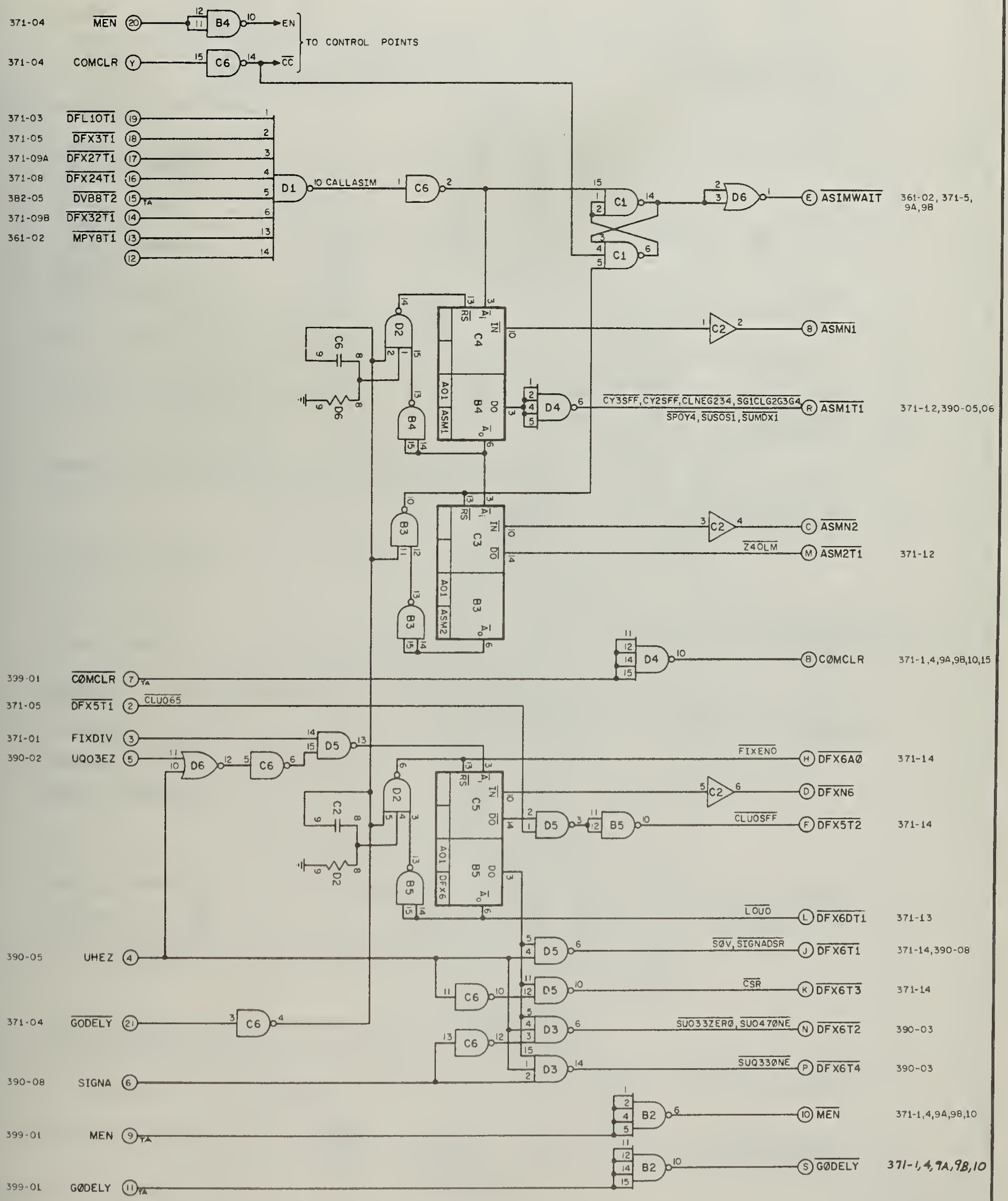
DEPARTMENT of COMPUTER SCIENCE University of Illinois			
For L. G.	Drawn by T. G	Date 12-28-71	Supersedes dwg.
Approved by	Date 1-11-72	Reference	

Issue	Change order	Approved by	Date

REVISIONS

TITLE AU DIVISION CONTROL LOGIC
'DIVIDE' AND QUOTIENT ASSIMILATION
DIVIDE AND DFL

Sheet_____ of _____ Drawing No. AUO-07-371-03



CARD LOCATION: H52

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

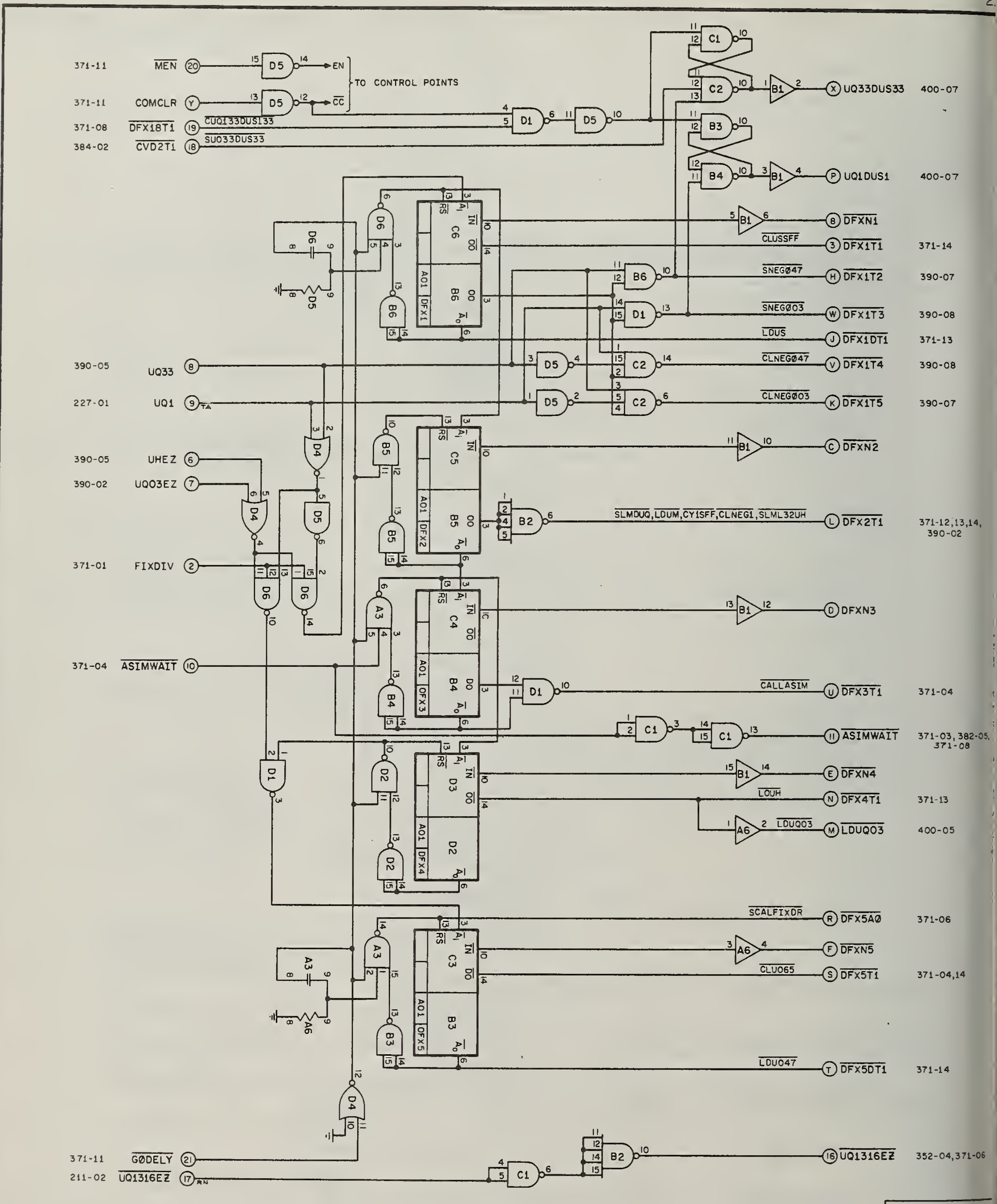
TITLE AU DIVISION (FL.PT.) CONTROL LOGIC
CONVERSION TO CONVENTIONAL REPRESENTATION
ASIM

Issue	Change order	Approved by	Date
4	NO 597	[illegible]	5-26-72
3	NO. 542	[illegible]	3-30-72
2	NO 535	[illegible]	3-30-72

REVISIONS

For	Drawn by	Date	Supersedes dwg.
L G	T G	1-20-72	
Approved by	Date	Reference	
[illegible]	1 21 1972		

Sheet ____ of ____ Drawing No. AUO-07-371-04



CARD LOCATION: H30

2	NO. 598		
Issue	Change order	Approved by	Date

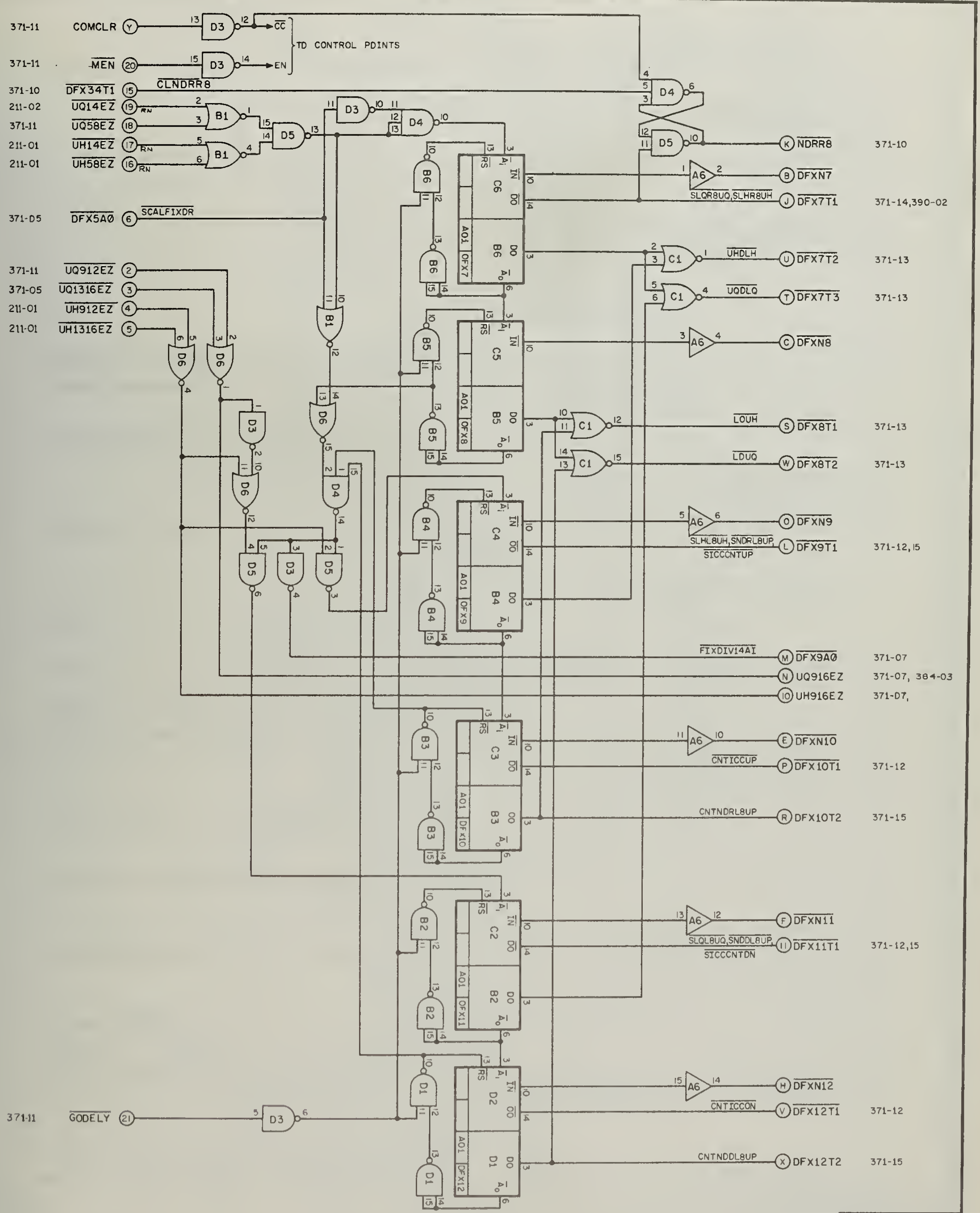
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L.G.	T.G.	1-19-72	
Approved by	Date	Reference	
	1 19 72		

TITLE AU DIVISION (FL. PT.) CONTROL LOGIC
DIVISOR/DIVIDEND ZERO TEST AND
2'S COMPLEMENTATION
DIZETCOM

Sheet ____ of ____ Drawing No. AUO-07-371-05



CARD LOCATION:
H-28

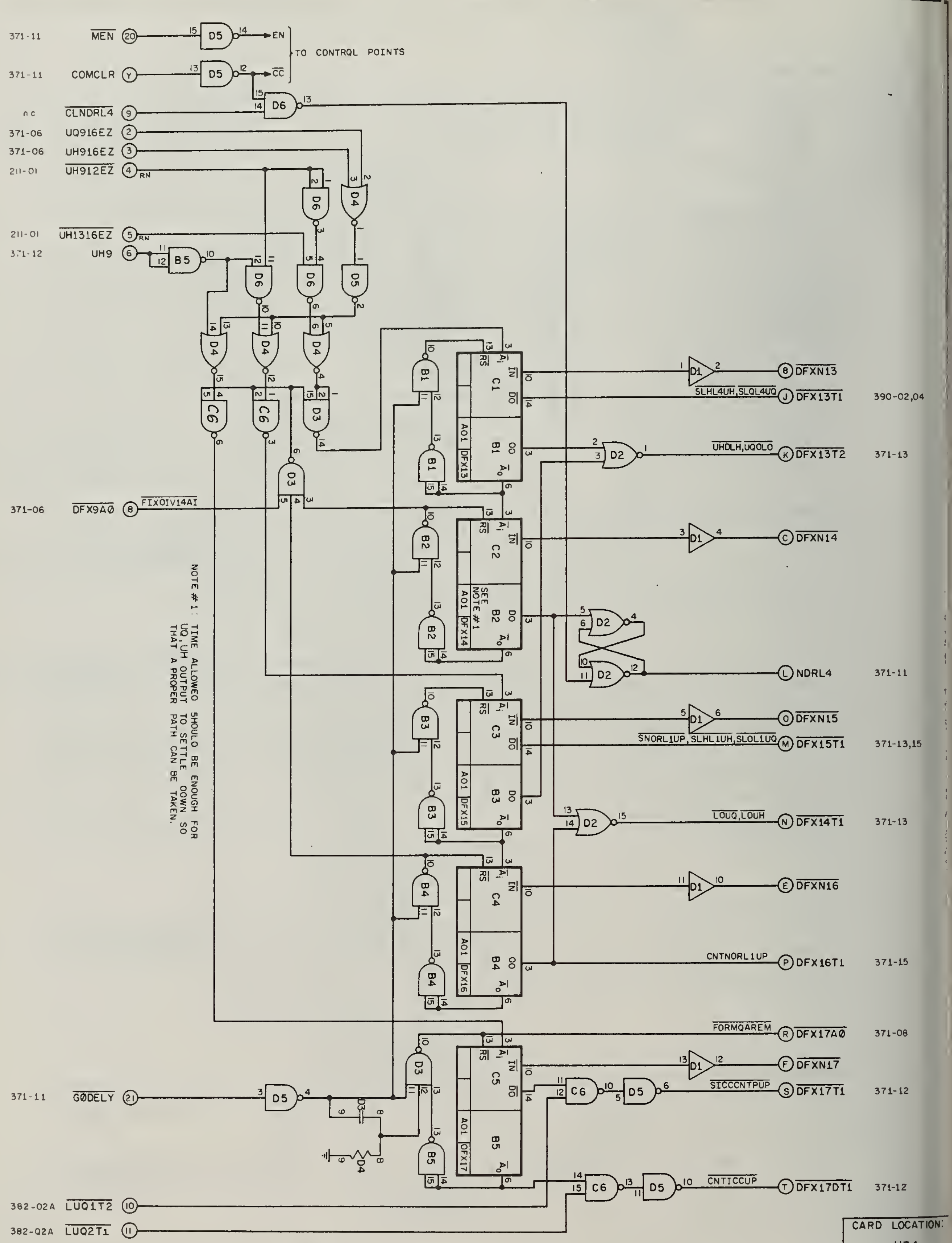
Issue	Change order	Approved by	Date
REVISIONS			

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg
L.G.	T.G	1-18-72	
Approved by	Date	Reference	

TITLE AU DIVISION (FL. PT.) CONTROL LOGIC
DIVISOR SCALING TO ≥1/2
SCALFIXDR

Sheet ____ of ____ Drawing No. AU0-07-371-06



CARD LOCATION: H34

Issue	Change order	Approved by	Date
5	No.629	P Krabbe	1-30-73
4	NO.599	P Krabbe	5-25-72
3	No.582	P Krabbe	3-30-72
2	No.541	P Krabbe	3-30-72

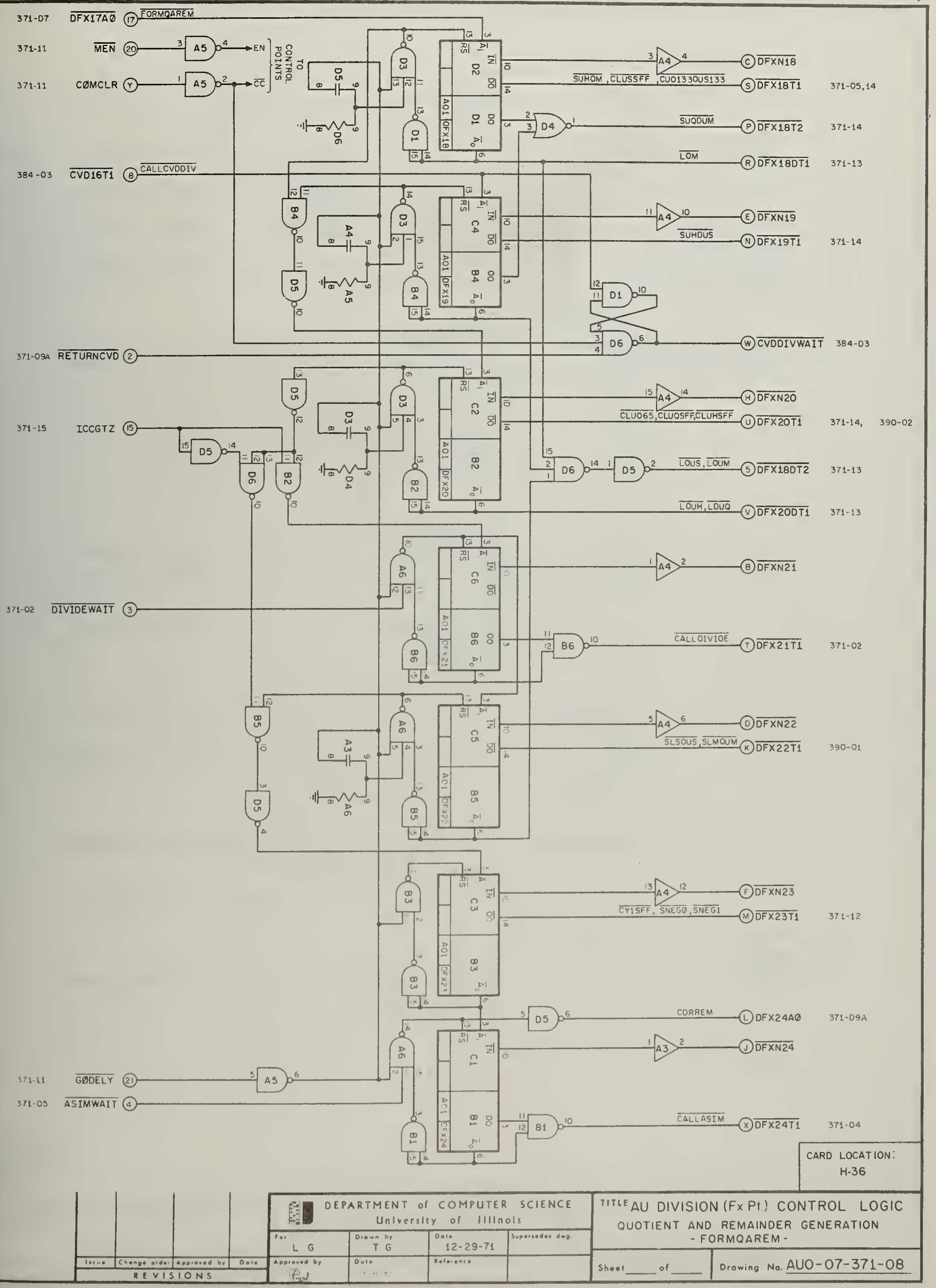
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

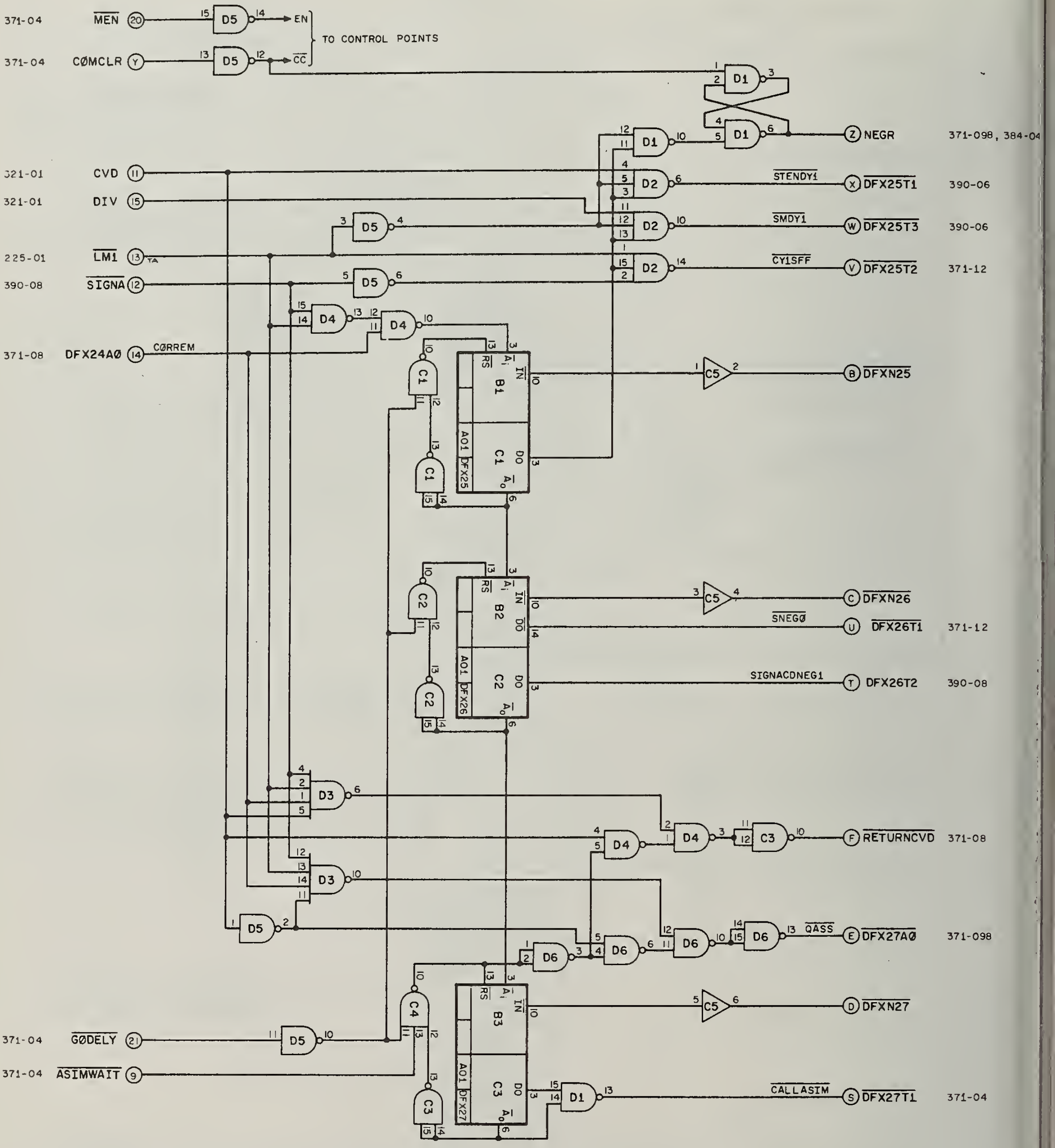
For	Drawn by	Date	Supersedes dwg.
L.G.	T. G.	1-19-72	
Approved by	Date	Reference	
	1 20 1972		

TITLE AU DIVISION (FL.PT.) CONTROL LOGIC
DIVISOR SCALING TO >1/2
SCALFIXDR

Sheet ____ of ____ Drawing No. AUO-07-371-07



371-07 DFX17A0 (17) FORMQAREM
371-11 MEN (20)
371-11 COMCLR (Y)
EN
CC
TO CONTROL POINTS
384-03 CVD16T1 (8) CALLCVDDIV
371-09A RETURNCVD (2)
371-15 ICCGTZ (15)
371-02 DIVIDEWAIT (3)
371-11 GODELY (21)
371-05 ASIMWAIT (4)
(C) DFXN18
SUHOM, CLUSSFF, CU0133OUS133
(S) DFX18T1 371-05,14
SUQDUM
(P) DFX18T2 371-14
LOM
(R) DFX18DT1 371-13
(E) DFXN19
SUHDUS
(N) DFX19T1 371-14
(W) CVDDIVWAIT 384-D3
(H) DFXN20
CLU065, CLUQSFF, CLUHSFF
(U) DFX20T1 371-14, 390-02
LOUS, LOUM
(5) DFX18DT2 371-13
LOUH, LDUQ
(V) DFX20DT1 371-13
(B) DFXN21
CALLDIVIDE
(T) DFX21T1 371-02
(D) DFXN22
SLSOUS, SLMOUM
(K) DFX22T1 390-01
(F) DFXN23
CY1SFF, SNEG0, SNEG1
(M) DFX23T1 371-12
CORREM
(L) DFX24A0 371-09A
(J) DFXN24
CALLASIM
(X) DFX24T1 371-04
CARD LOCATION: H-36
DEPARTMENT of COMPUTER SCIENCE
University of Illinois
For L G
Drawn by T G
Date 12-29-71
Supersedes dwg.
Approved by
Date
Reference
Issue | Change order | Approved by | Date
REVISIONS
TITLE AU DIVISION (Fx Pt.) CONTROL LOGIC
QUOTIENT AND REMAINDER GENERATION
- FORMQAREM -
Sheet ___ of ___
Drawing No. AUO-07-371-08



CARD LOCATION: H-58

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

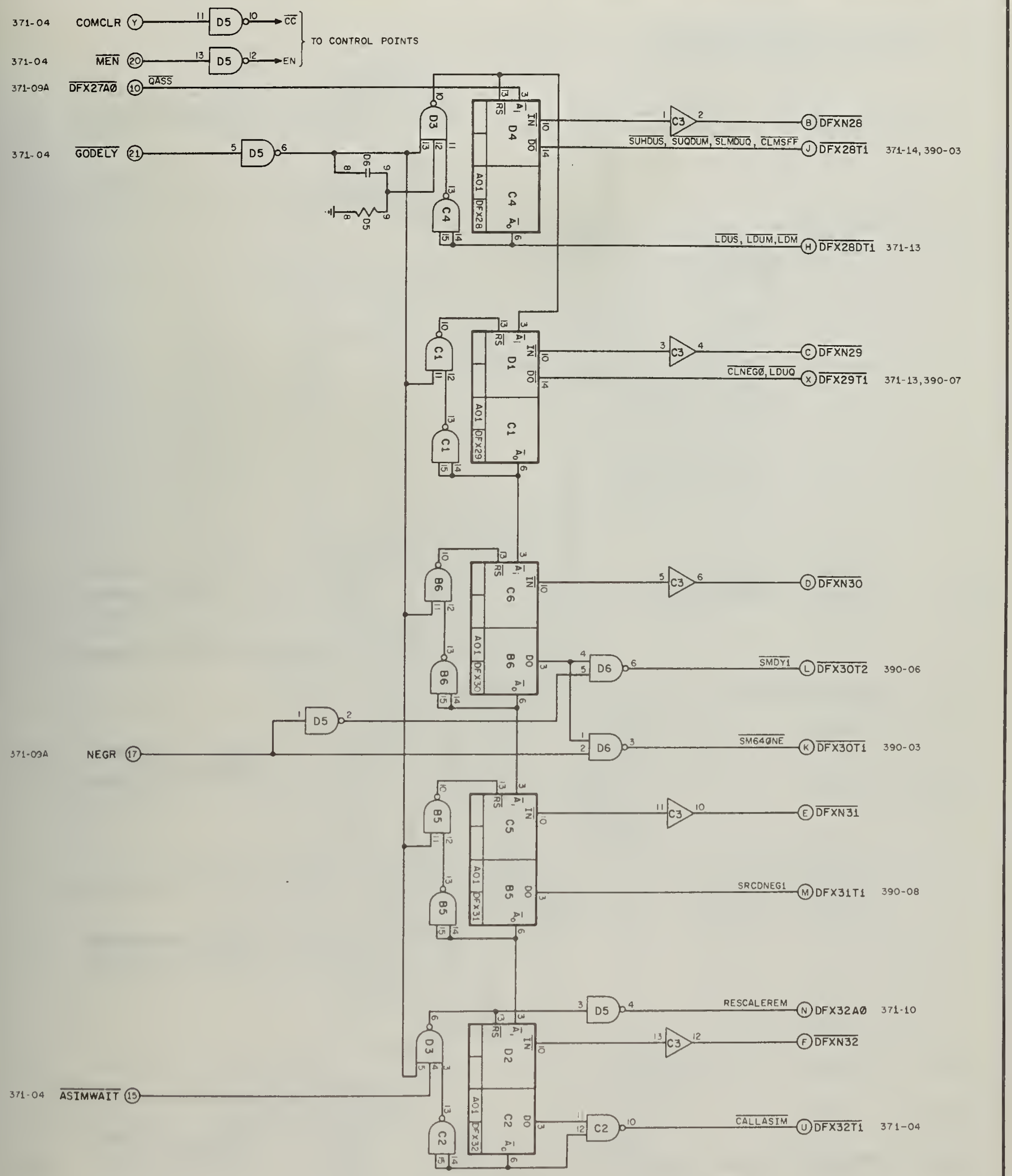
TITLE AU DIVISION (Fx.Pt.) CONTROL LOGIC
REMAINDER'S SIGN CORRECTION
- COSREM -

For	Drawn by	Date	Supersedes dwg.
L G	T G.	11-24-71	
Approved by	Date	Reference	
	1-11-72		

Sheet ____ of ____ Drawing No. AUO-07-371-09

Issue	Change order	Approved by	Date
2	No 534	P. Kraska	3-30-72

REVISIONS



CARD LOCATION:
H-56

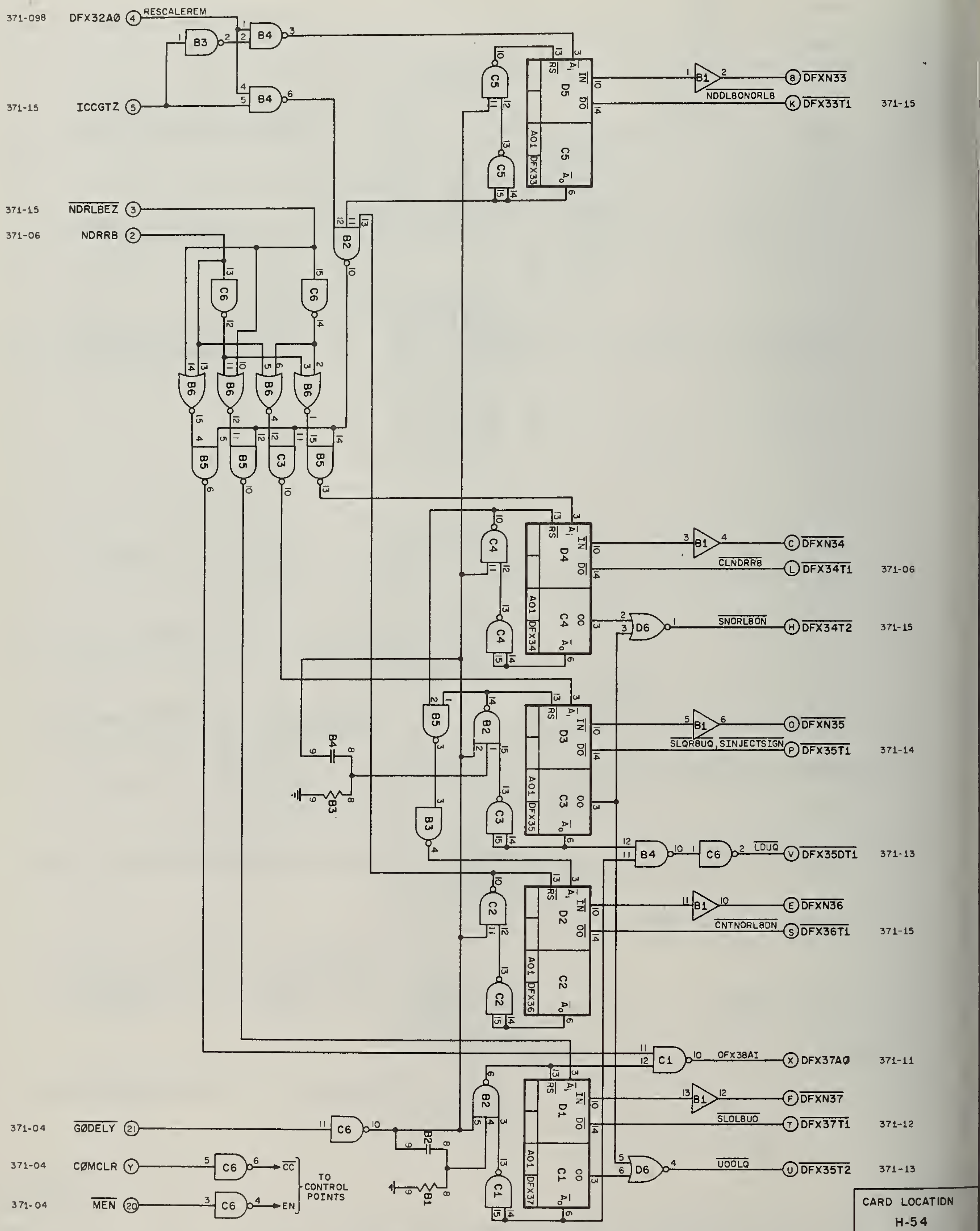
Issue	Change order	Approved by	Date
REVISIONS			

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L G	T G	11-22-71	
Approved by	Date	Reference	

TITLE AU DIVISION(Fx Pt) CONTROL LOGIC
QUOTIENT ASSIMILATION
- QASS -

Sheet ____ of ____ Drawing No. AUO-07-371-09B



CARD LOCATION
H-54

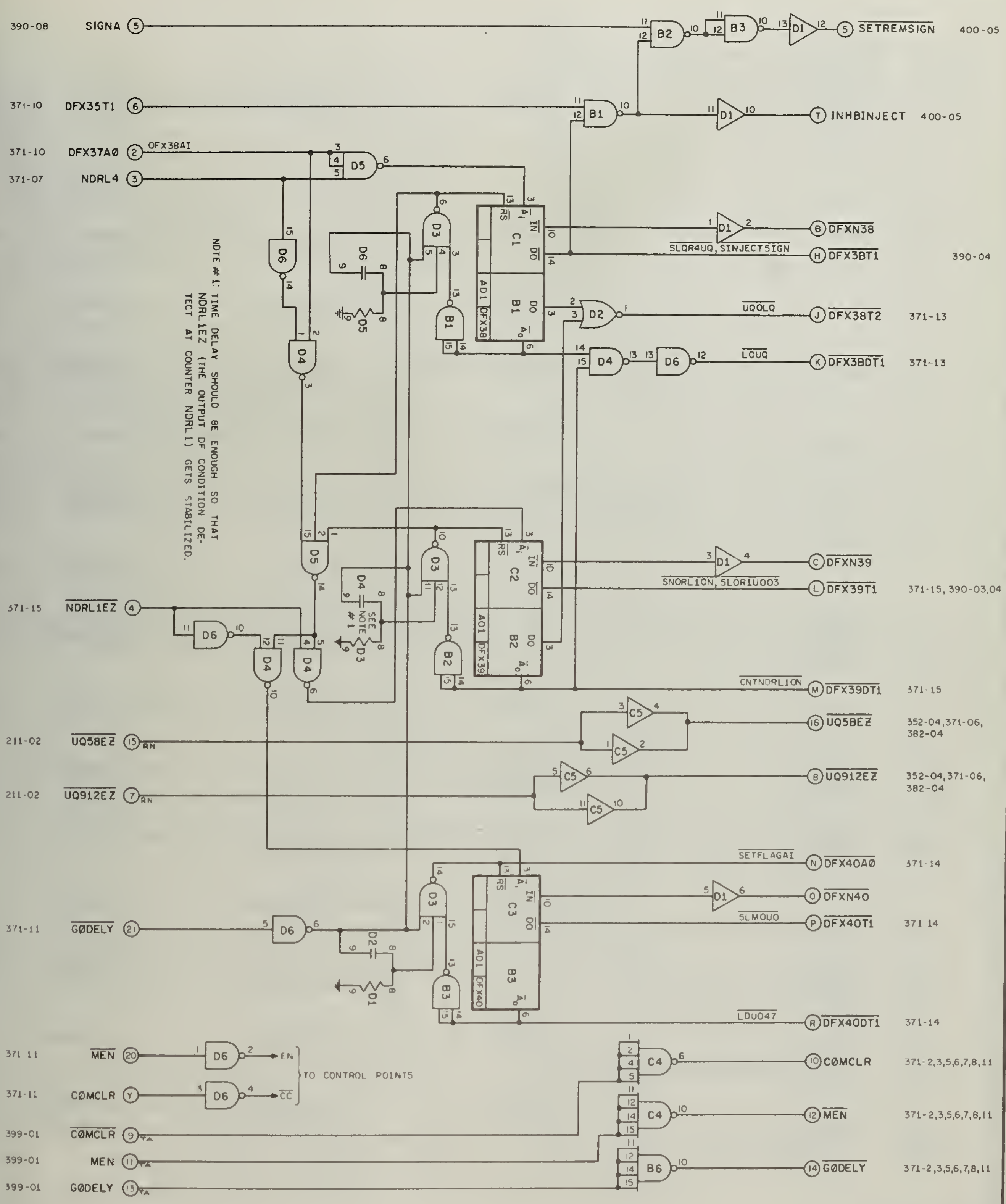
Issue	Change order	Approved by	Date
REVISIONS			

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L.G.	T.G.	11-29-71	
Approved by	Date	Reference	
	1-11-72		

TITLE AU DIVISION (Fx.Pt.) CONTROL LOGIC
REMAINDER POST-SCALING
- RESCALEREM -

Sheet ____ of ____ Drawing No. AUO-07-371-10



CARD LOCATION:
H32

Issue	Change order	Approved by	Date
5	No 615	P. Krabbe	8-10 72
4	No 600	P. Krabbe	5 25 72
3	No 578	P. Krabbe	3 30 72
2	No 536	P. Krabbe	2 30 72

REVISIONS

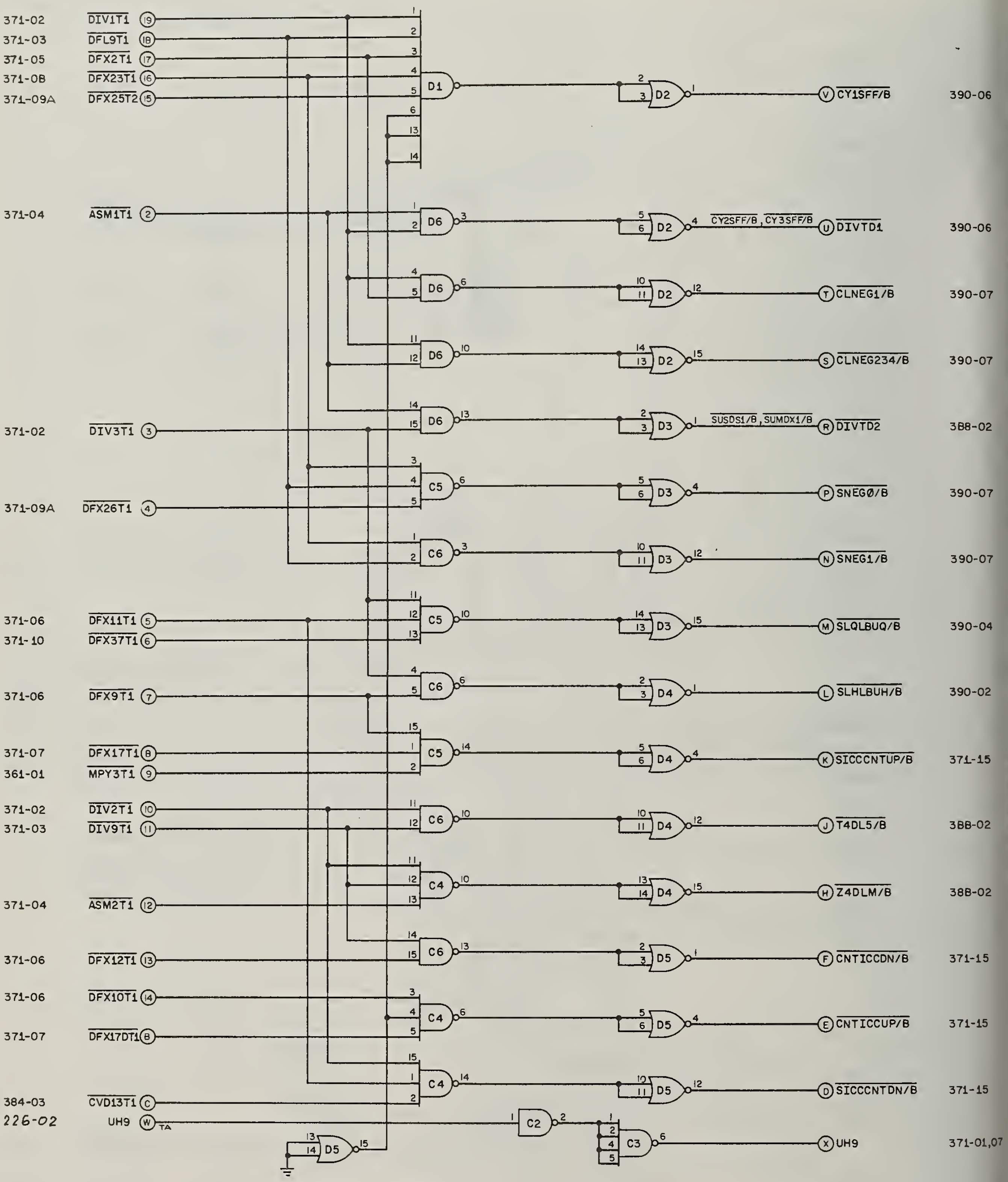
DEPARTMENT of COMPUTER SCIENCE
University of Illinois

TITLE AU - DIVISION (FL.PT) CONTROL LOGIC
REMAINDER POST-SCALING
RESCALEREM

For	Drawn by	Date	Supersedes dwg
L. G.	T G	1-26-71	

Approved by	Date	Reference
	1 27 872	

Sheet ____ of ____ Drawing No. AUO-07-371-11



CARD LOCATION:
H44

Issue	Change order	Approved by	Date
3	NO. 601	P Krahily	5-25-72
2	No 556	P Krahily	3-30-72

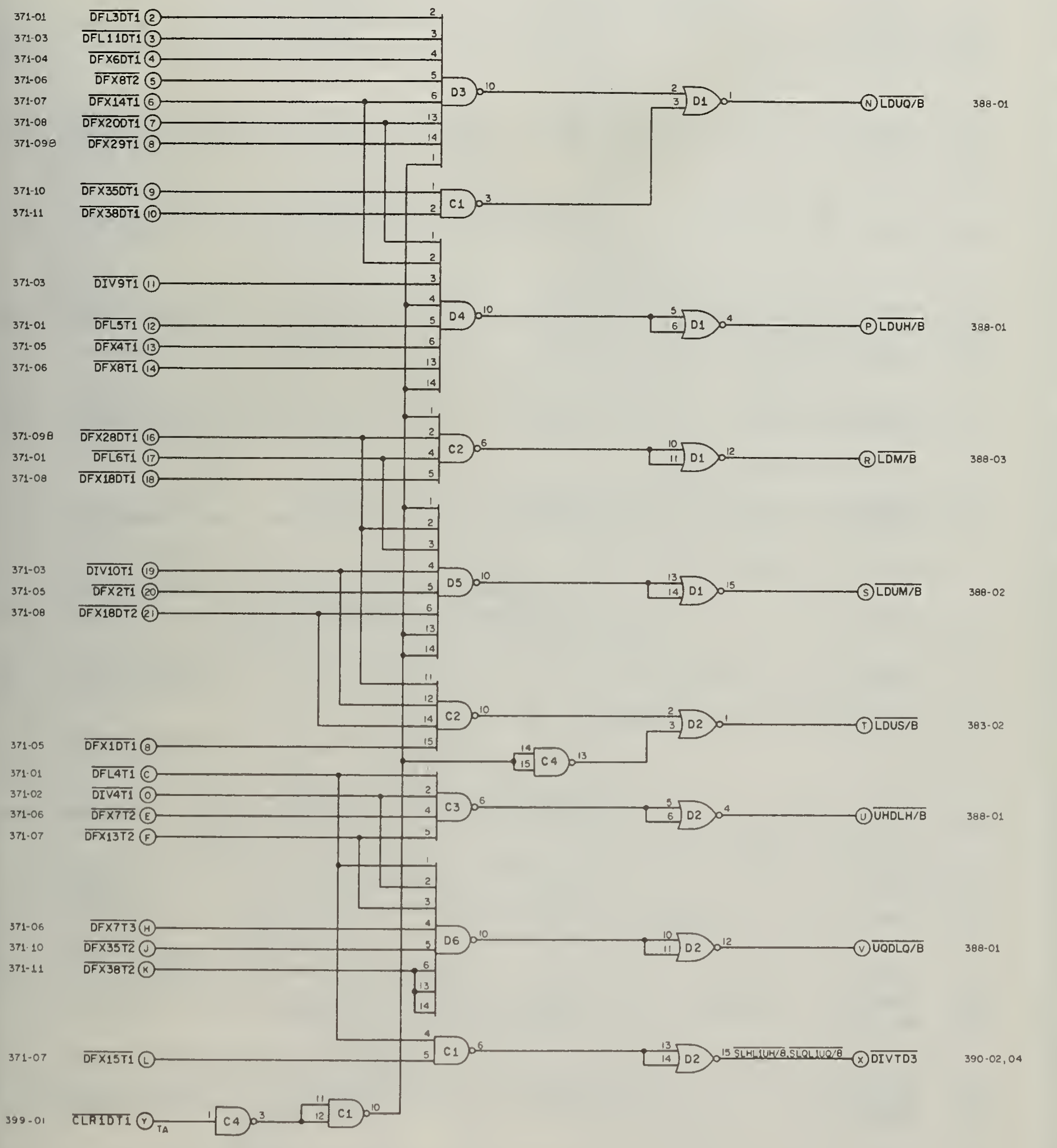
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L. G.	T. G.	11-1-71	
Approved by	Date	Reference	
Gmal	1-11-72		

TITLE AU DIVISION (Fx Pt) CONTROL LOGIC
TASK SIGNAL COLLECTOR "B1"

Sheet ____ of ____ Drawing No. AUO-07-371-12



CARD LOCATION:
H-46

Issue	Change order	Approved by	Date
4	No 630	P Krabbe	1-3-73
3	No 557	P Krabbe	3-30-72
2	No 540	P Krabbe	3-30-72

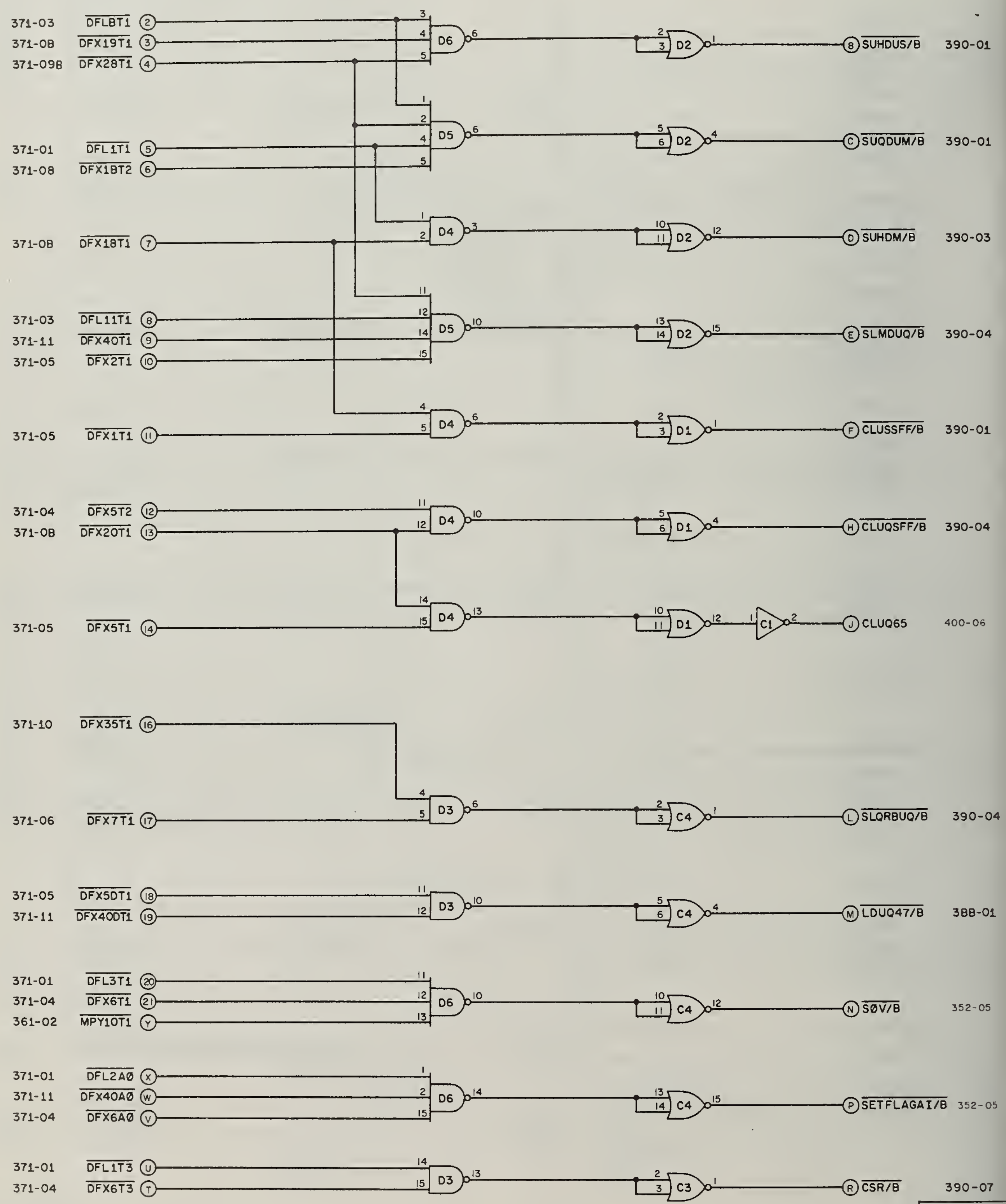
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L.G	T G	11-5-71	
Approved by	Date	Reference	
[signature]	1-11-72		

TITLE AU DIVISION (Fx.Pt.) CONTROL LOGIC
TASK SIGNAL COLLECTOR "B2"

Sheet ____ of ____ Drawing No. AUO-07-371-13



CARD LOCATION:
H-40

Issue	Change order	Approved by	Date
2	No 555	P Krabbe	3-30-72

REVISIONS

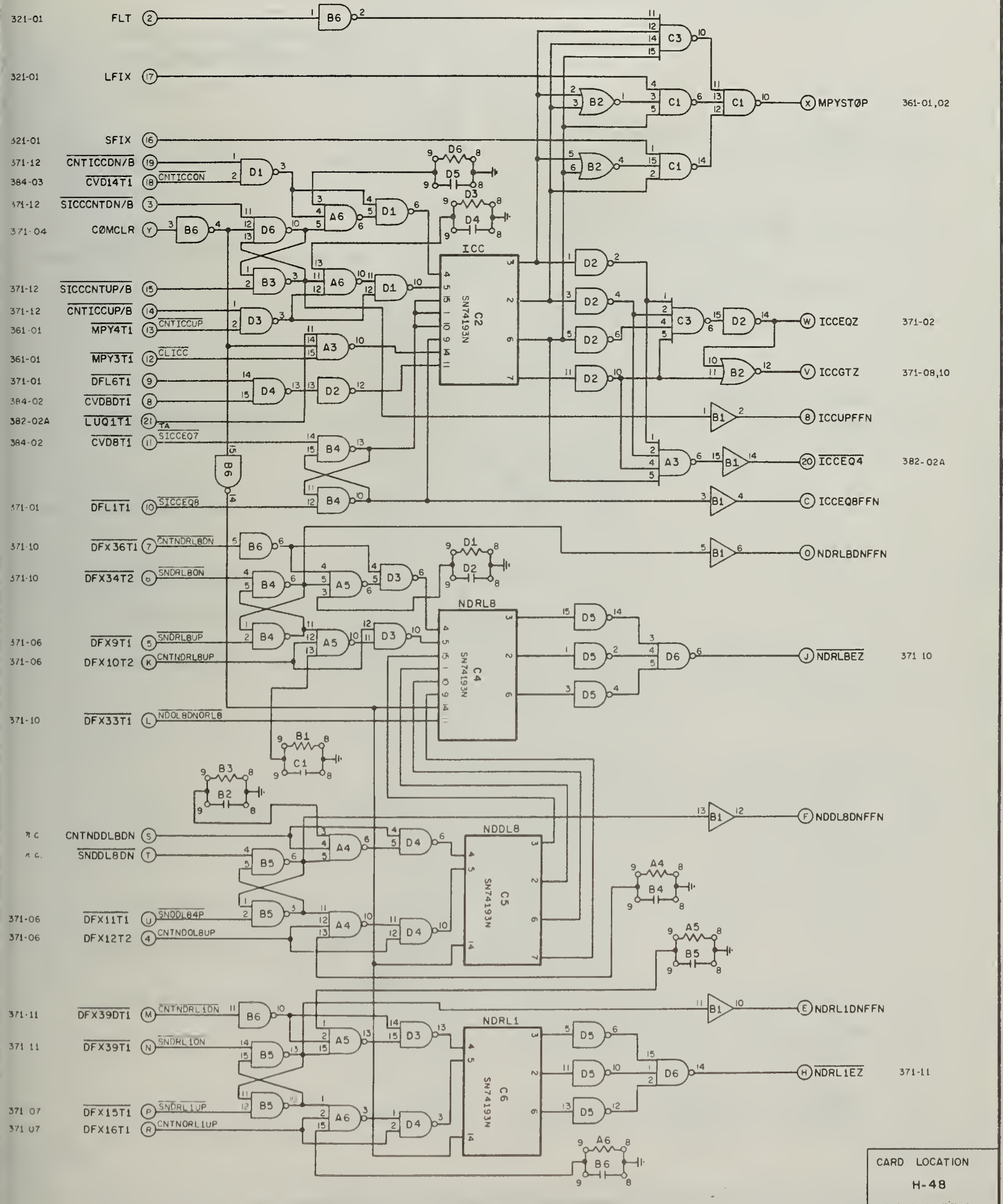
DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L.G.	T. G	11-3-71	
Approved by	Date	Reference	
[signature]	1-41-72		

TITLE AU DIVISION (Fx. Pt.) CONTROL LOGIC
TASK SIGNAL COLLECTOR "B3"

Sheet ____ of ____

Drawing No. AUO-07-371-14



DEPARTMENT of COMPUTER SCIENCE
University of Illinois

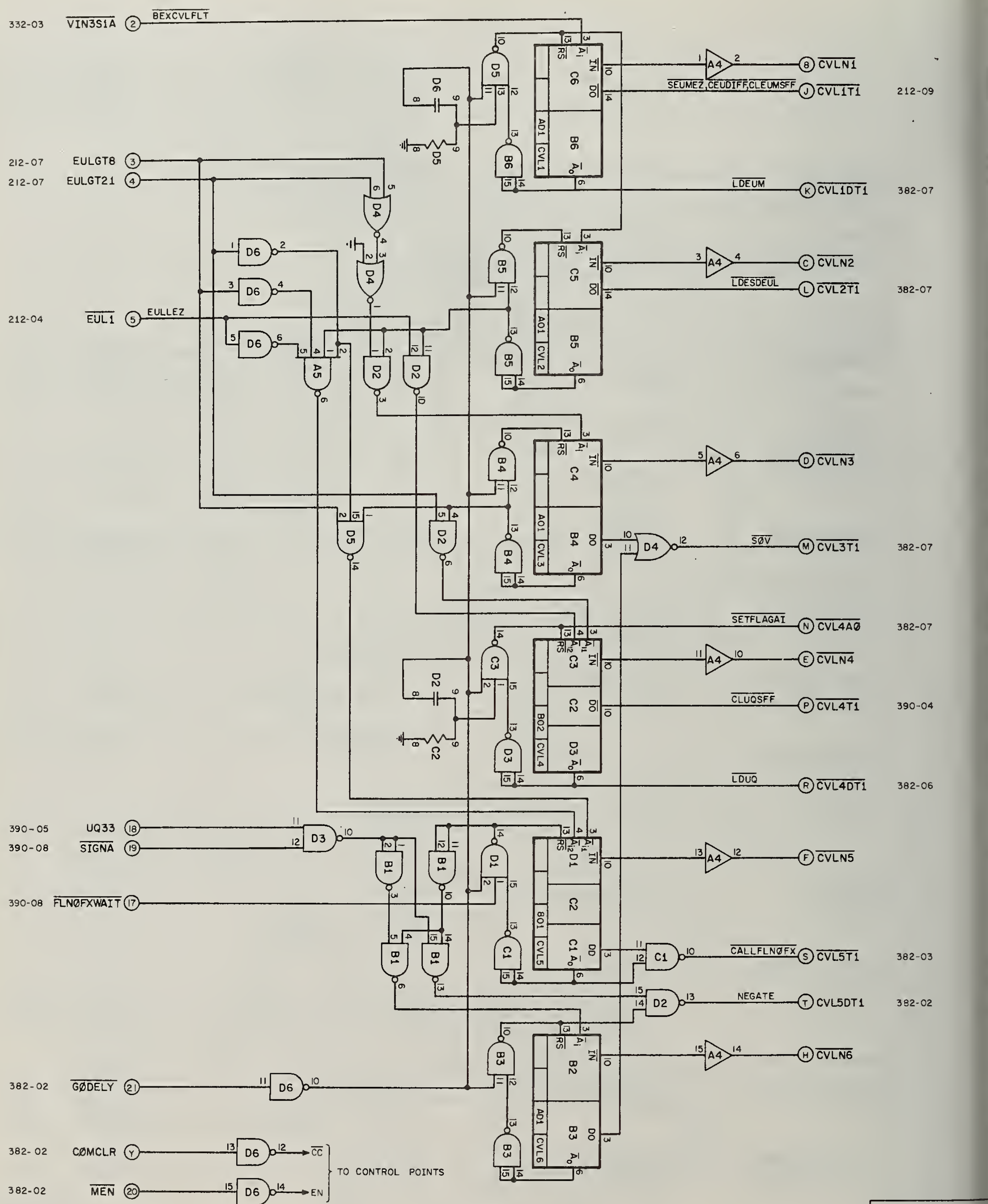
TITLE AU DIVISION (Fx Pt) CONTROL LOGIC
COUNTERS ICC, NDRL8, NDDL8, NDRL1

Issue	Change order	Approved by	Date
2	NO 631		1-30-73

REVISIONS

For: L G — Drawn by: T G — Date: 12-27-71 — Supersedes dwg
Approved by — Date: 1-11-72 — Reference

Sheet ____ of ____ Drawing No. AUO-07-371-15



CARD LOCATION:
E-28

2	NO. 537	P. Kraska	3-30-72
Issue	Change order	Approved by	Date

REVISIONS

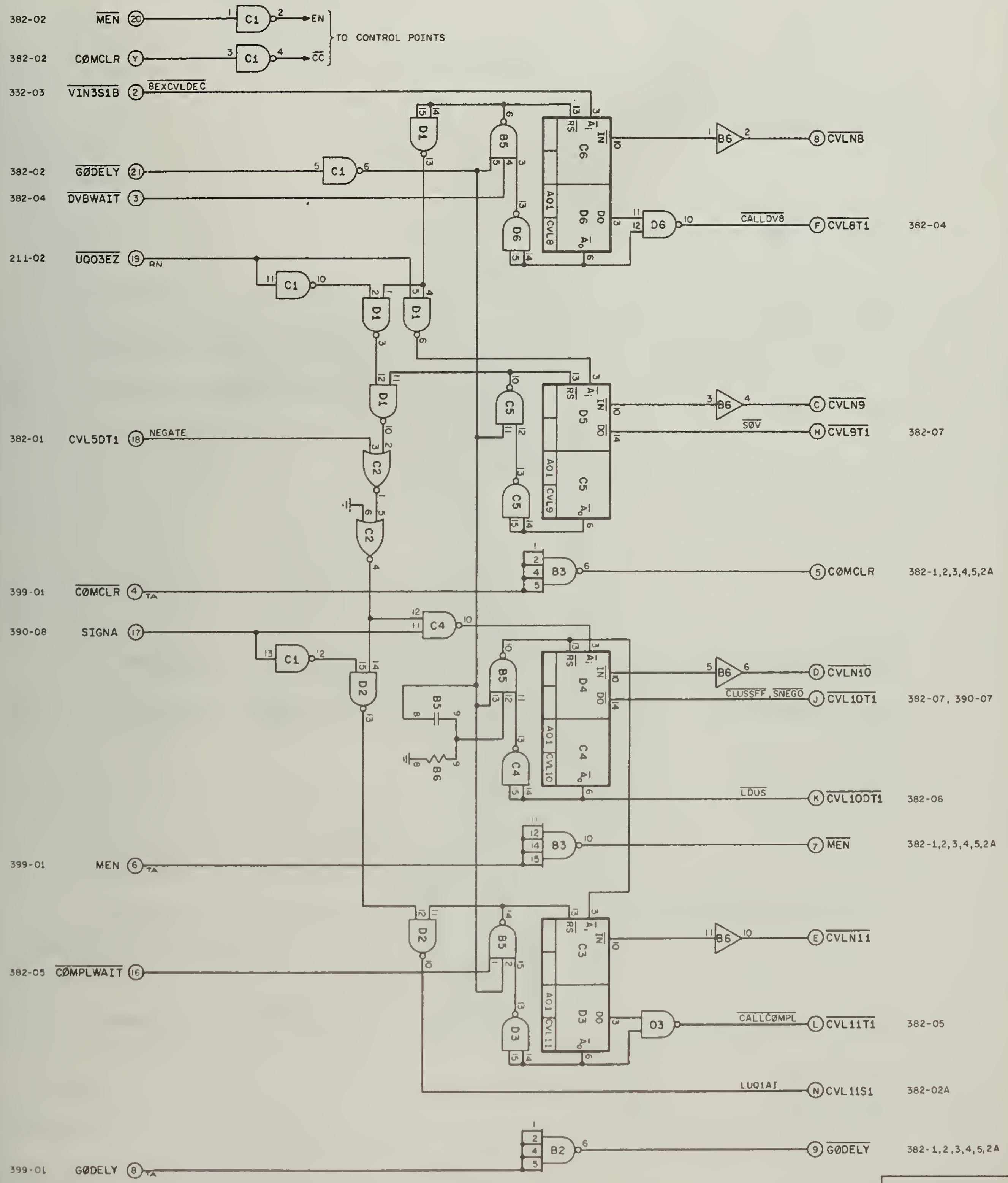
DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L. G.	T. G.	12-21-71	
Approved by	Date	Reference	
Gyel	1-11-72		

TITLE AU - CVL CONTROL LOGIC
FLOATING POINT OPERAND
CVL_FLT

Sheet ____ of ____

Drawing No. AUO-07-382-01



CARD LOCATION:
E24

Issue	Change order	Approved by	Date
3	NO 832	P. Malle	1-30-73
2	NO 602	P. Kraskle	3-25-72

REVISIONS

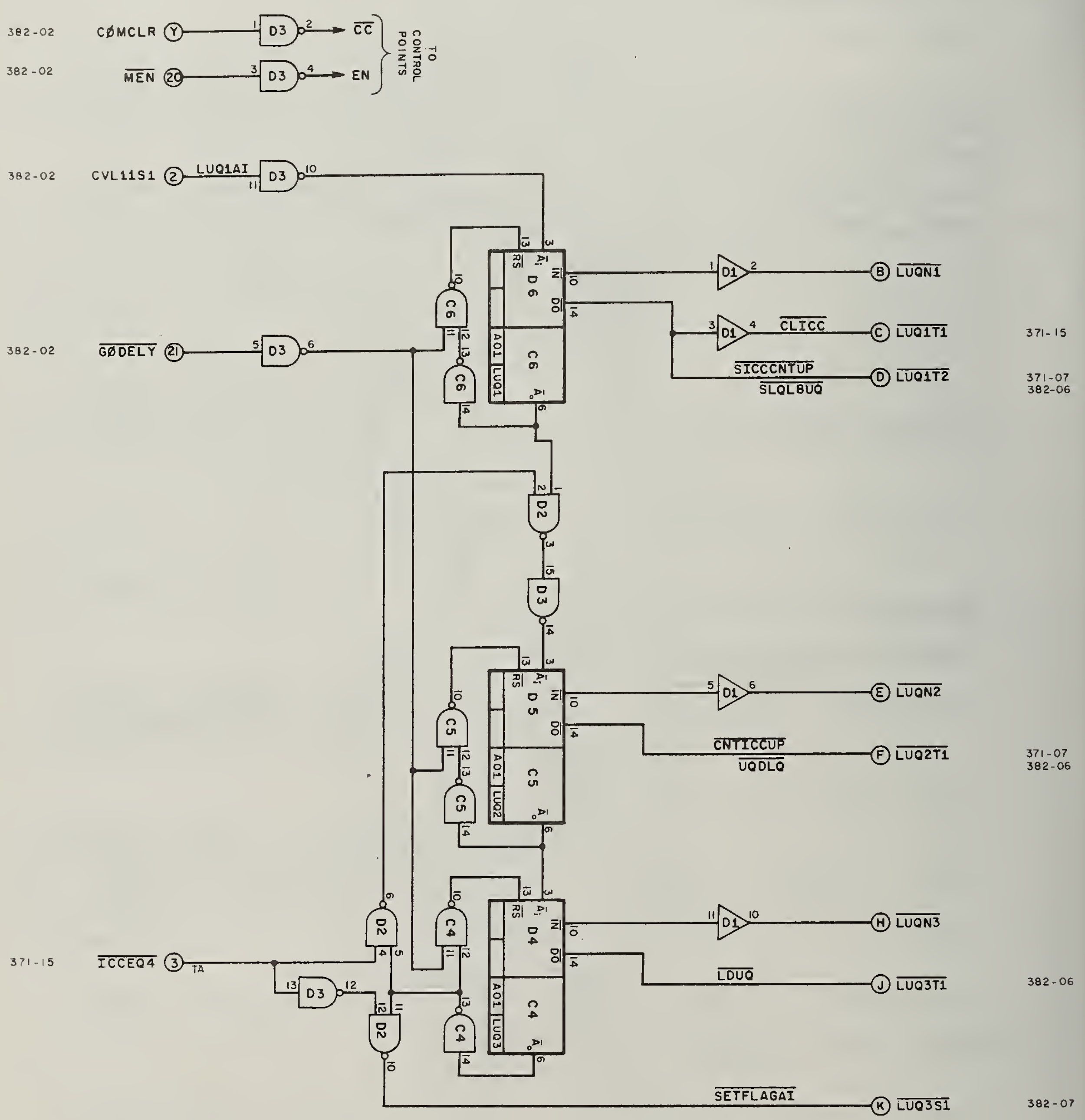
DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For: L. G. | Drawn by: T G | Date: 11-30-71 | Supersedes dwg.
Approved by: Pool. | Date: 1-11-72 | Reference

TITLE AU - CVL CONTROL LOGIC
DECIMAL OPERAND
CVL_DEC

Sheet ____ of ____

Drawing No. AUO-07-382-02



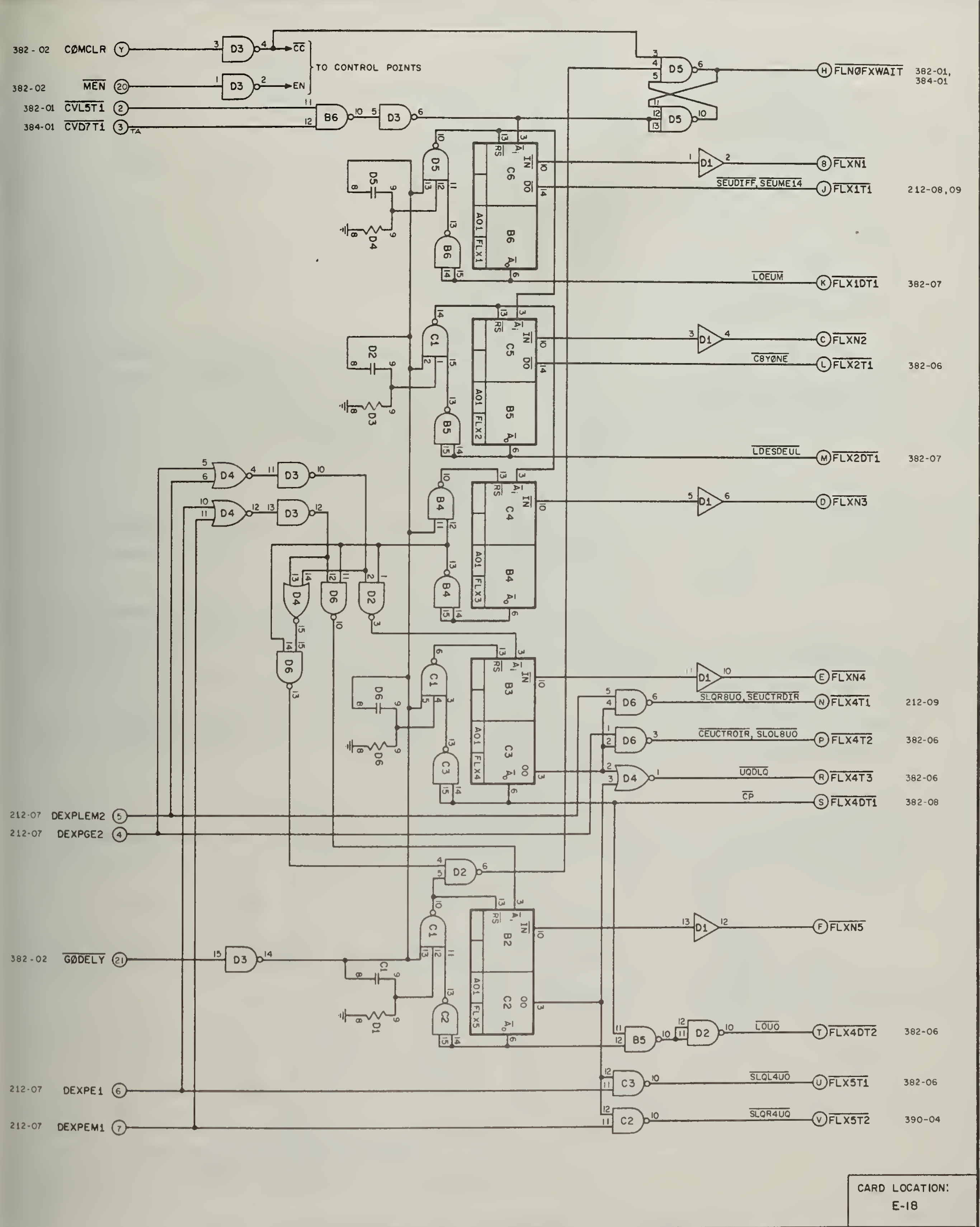
CARD LOCATION: E 30

Issue	Change order	Approved by	Date
REVISIONS			

DEPARTMENT of COMPUTER SCIENCE University of Illinois			
For PK	Drawn by B.W.	Date 10-5-72	Supersedes dwg.
Approved by P. Krabbe	Date 10-24-72	Reference	

TITLE AU CVL CONTROL LOGIC LEFT ADJUST UQ -LUQ-

Sheet ____ of ____ Drawing No. AUO-07-382-02A



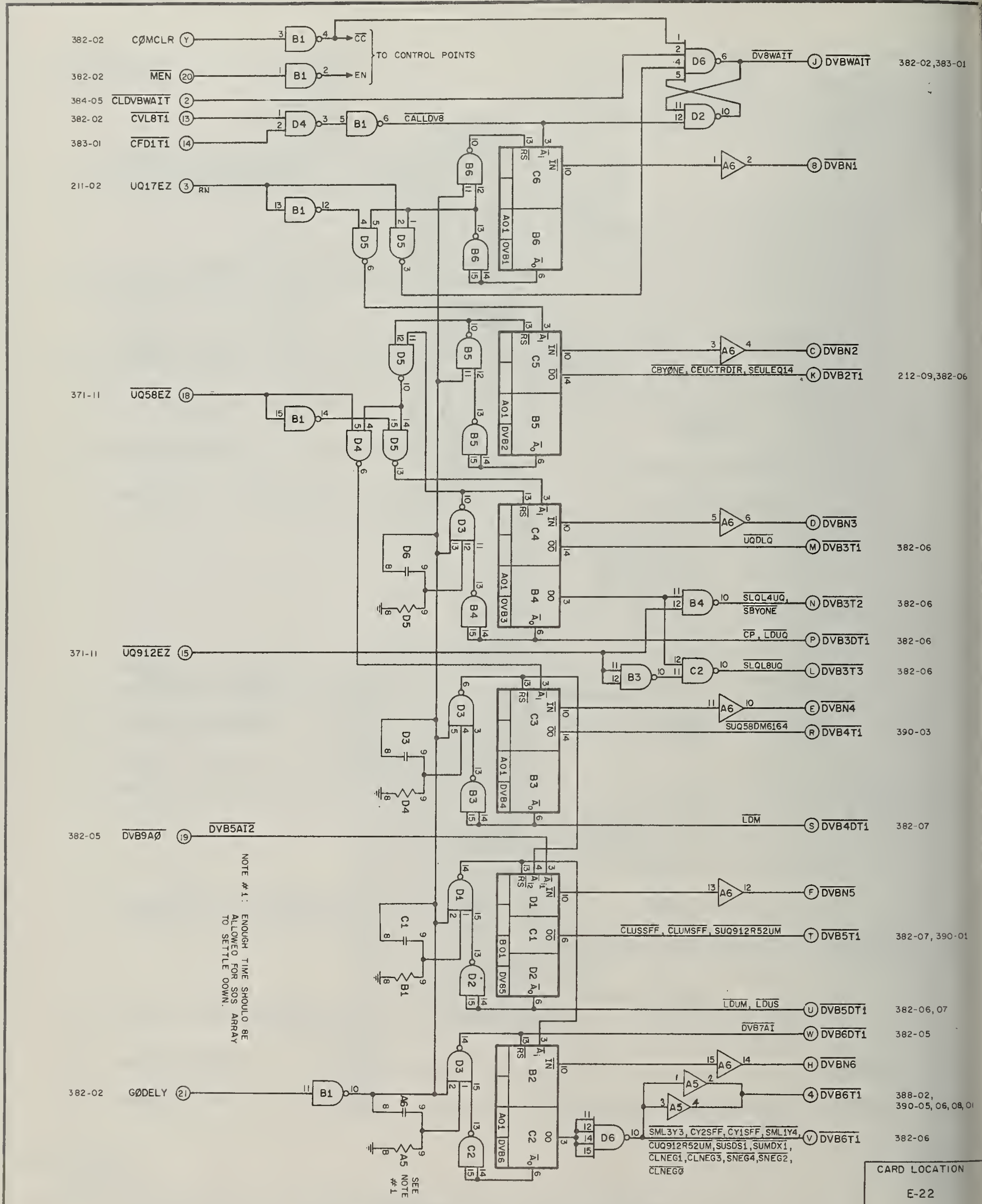
Issue	Change order	Approved by	Date
REVISIONS			

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L.G.	T.G.	11-30-71	
Approved by	Date	Reference	
	1-11-72		

TITLE AU-CVL CONTROL LOGIC
FLOATING-NORMALIZE-FIX
- FL-NØRM-FX -

Sheet ___ of ___ Drawing No. AUO-07-382-03



Issue	Change order	Approved by	Date
2	No 610	P. Frable	8-10-72

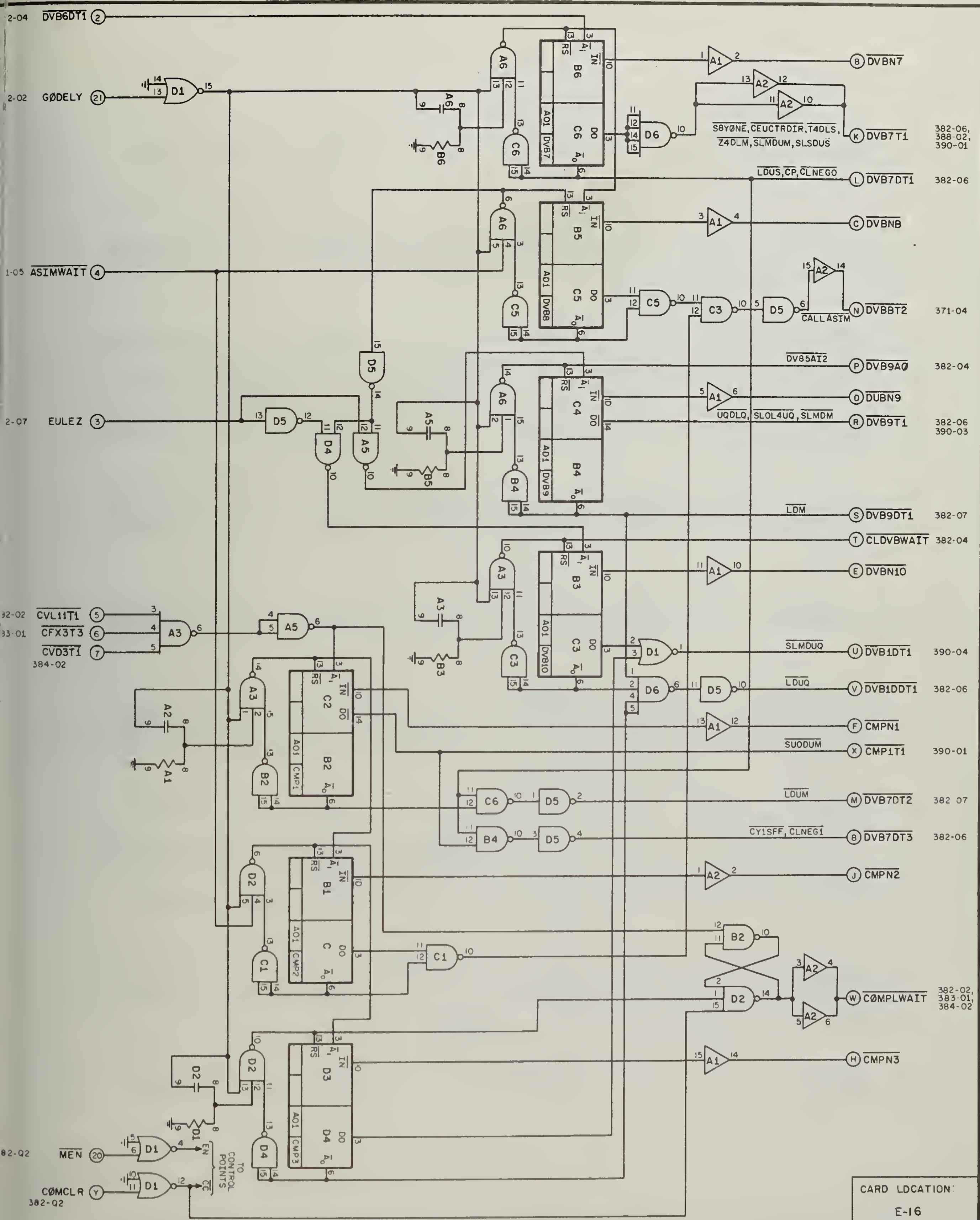
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L. G	T. G.	12-3-71	
Approved by	Date	Reference	
[illegible]	1-11-72		

TITLE AU - CVL CONTROL LOGIC
DECIMAL TO BINARY CONVERSION
- DVB -

Sheet ____ of ____ Drawing No AU0-07-382-04



CARD LOCATION: E-16

2	No 611	[illegible]	8 10 72
Issue	Change order	Approved by	Date

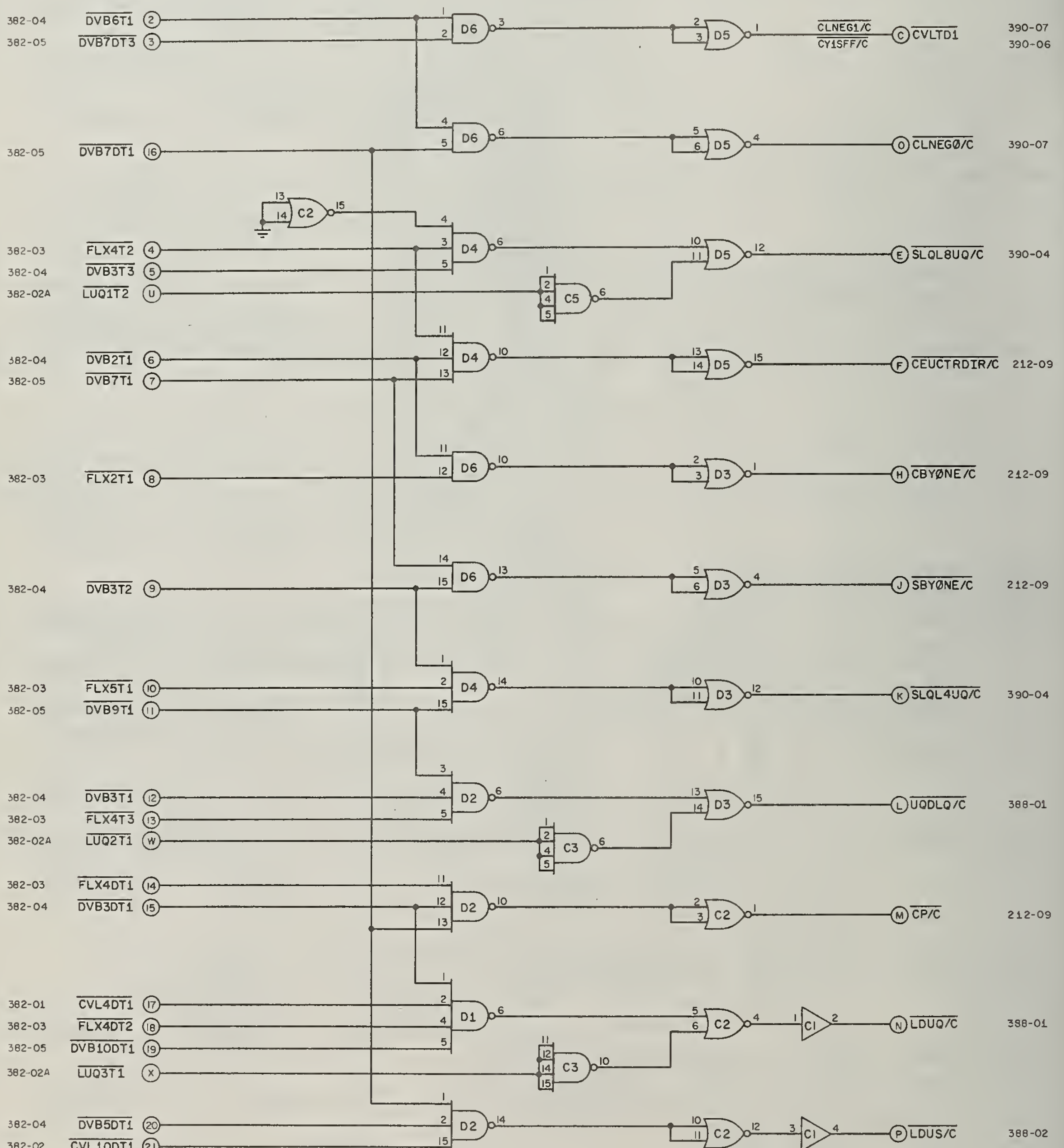
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For: L.G — Drawn by: T.G — Date: 11-17-71 — Supersedes dwg.
Approved by: [illegible] — Date: 1-11-72 — Reference

TITLE: AU - CVL CONTROL LOGIC
DECIMAL TO BINARY CONVERSION & COMPLEMENTATION
DVB AND COMPL

Sheet ____ of ____ Drawing No. AUO-07-382-05



CARD LOCATION: E-20

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

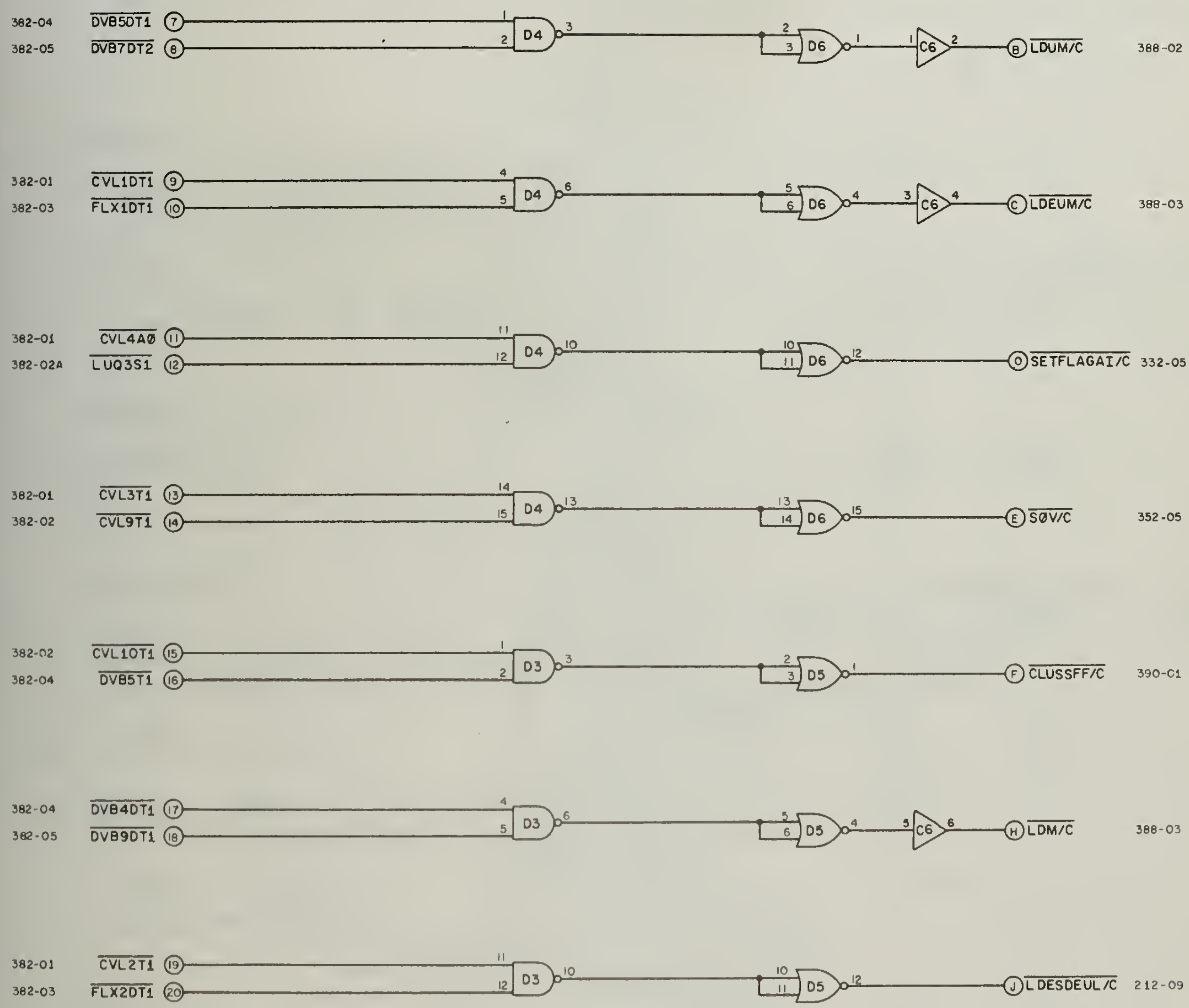
For	Drawn by	Date	Supersedes dwg.
L. G.	T. G.	11-5-71	
Approved by	Date	Reference	
[illegible]	1-11-72		

TITLE AU - CVL CONTROL LOGIC
TASK SIGNAL COLLECTOR "C1"

Sheet ____ of ____ Drawing No. AUO-07-382-06

Issue	Change order	Approved by	Date
3	633	[illegible]	1-30-73
2	619	[illegible]	8-10-72

REVISIONS



CARD LOCATION: E-26

Issue	Change order	Approved by	Date
2	No 620	P Kroldon	8-10-72

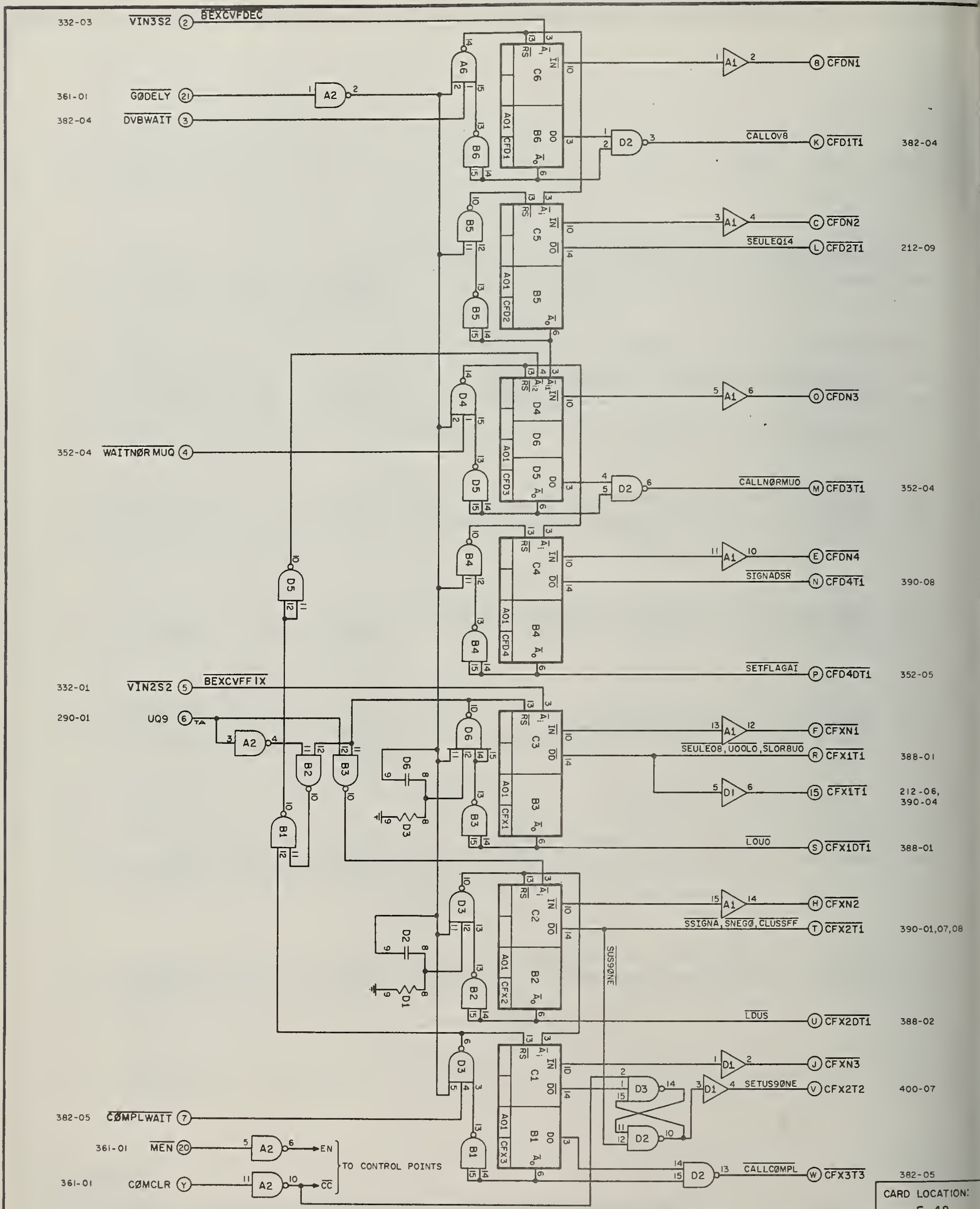
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L. G	T G	11-5-71	
Approved by	**Date**	**Reference**	
	1-11-72		

TITLE AU - CVL CONTROL LOGIC
TASK SIGNAL COLLECTOR "C2"

Sheet ____ of ____ Drawing No. AUO-07-382-07



CARD LOCATION: E-48

Issue	Change order	Approved by	Date
3	No. 612	P Knoffe	8-10-72
2	No. 579	P Knoffe	4-12-72

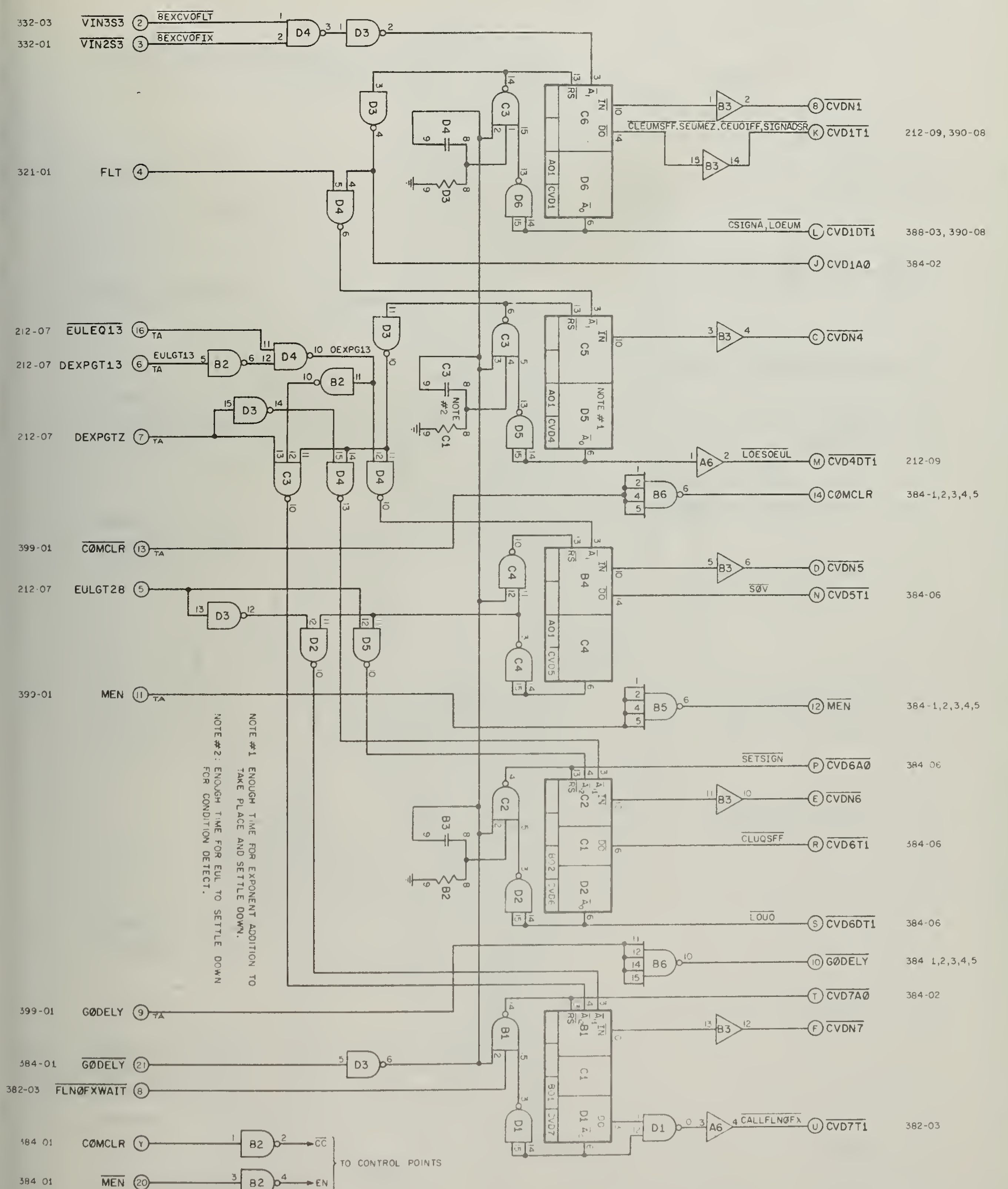
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For: L.G. | Drawn by: T. G. | Date: 12-8-71 | Supersedes dwg.:
Approved by: | Date: 1-11-72 | Reference:

TITLE AU-CVF CONTROL LOGIC
DECIMAL AND FIXED POINT OPERANDS
CVF_DEFIX

Sheet ____ of ____ Drawing No. AUO-07-383-01



CARD LOCATION
E60

Issue	Change order	Approved by	Date
4	NO 634		1-30 73
3	613		8-10-72
2	NO 603		4-21-72

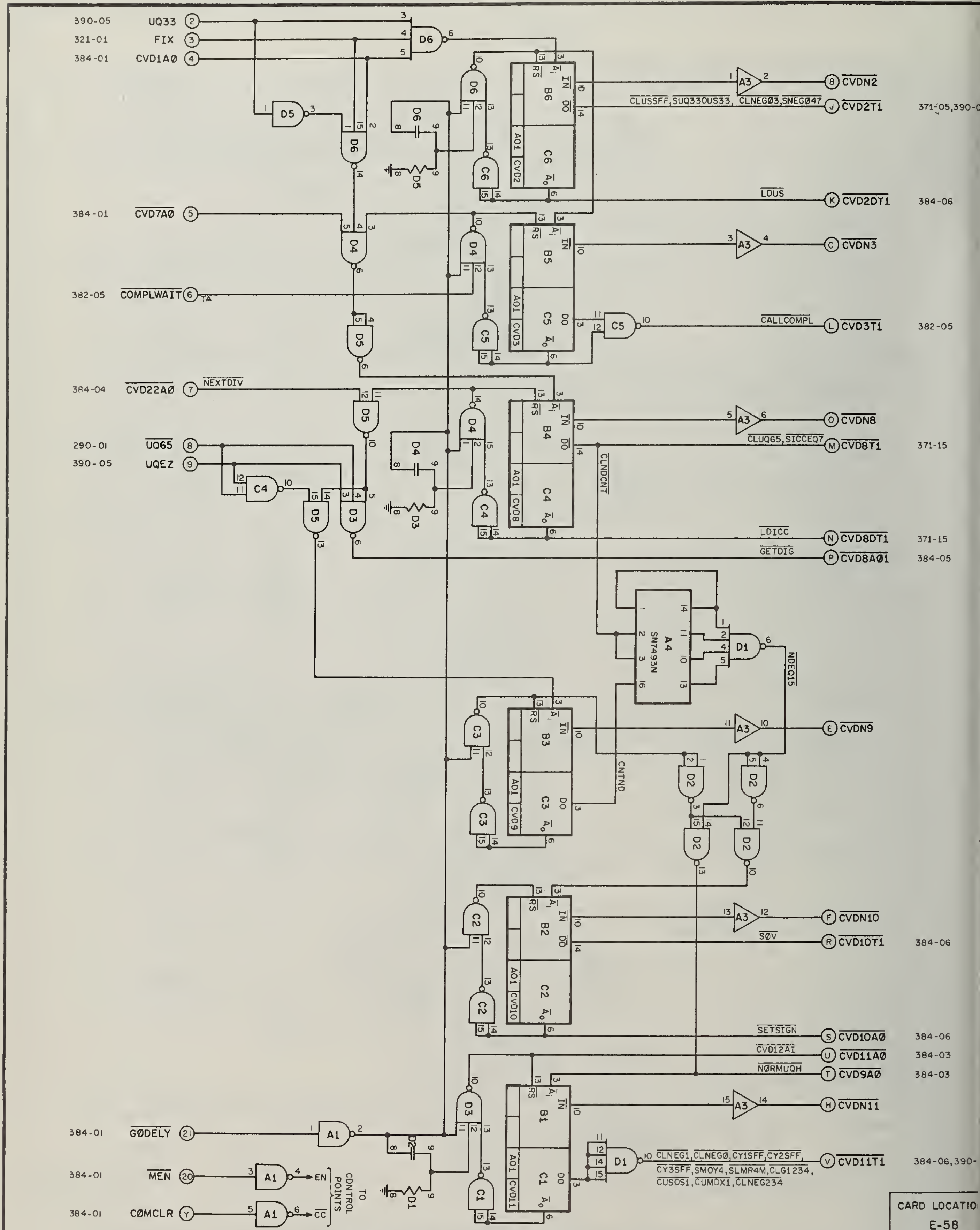
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L.G	TG	12-20-71	
Approved by	Date	Reference	
	1-4-72		

TITLE AU - CVD CONTROL LOGIC
Fx Pt OPERAND COMPLEMENTATION AND Fl Pt → Fx Pt
- FIXOCAFLOX -

Sheet ____ of ____ Drawing No. AUO-07-384-01



2	NO 538	P. Krabbe	3-30-72
Issue	Change order	Approved by	Date

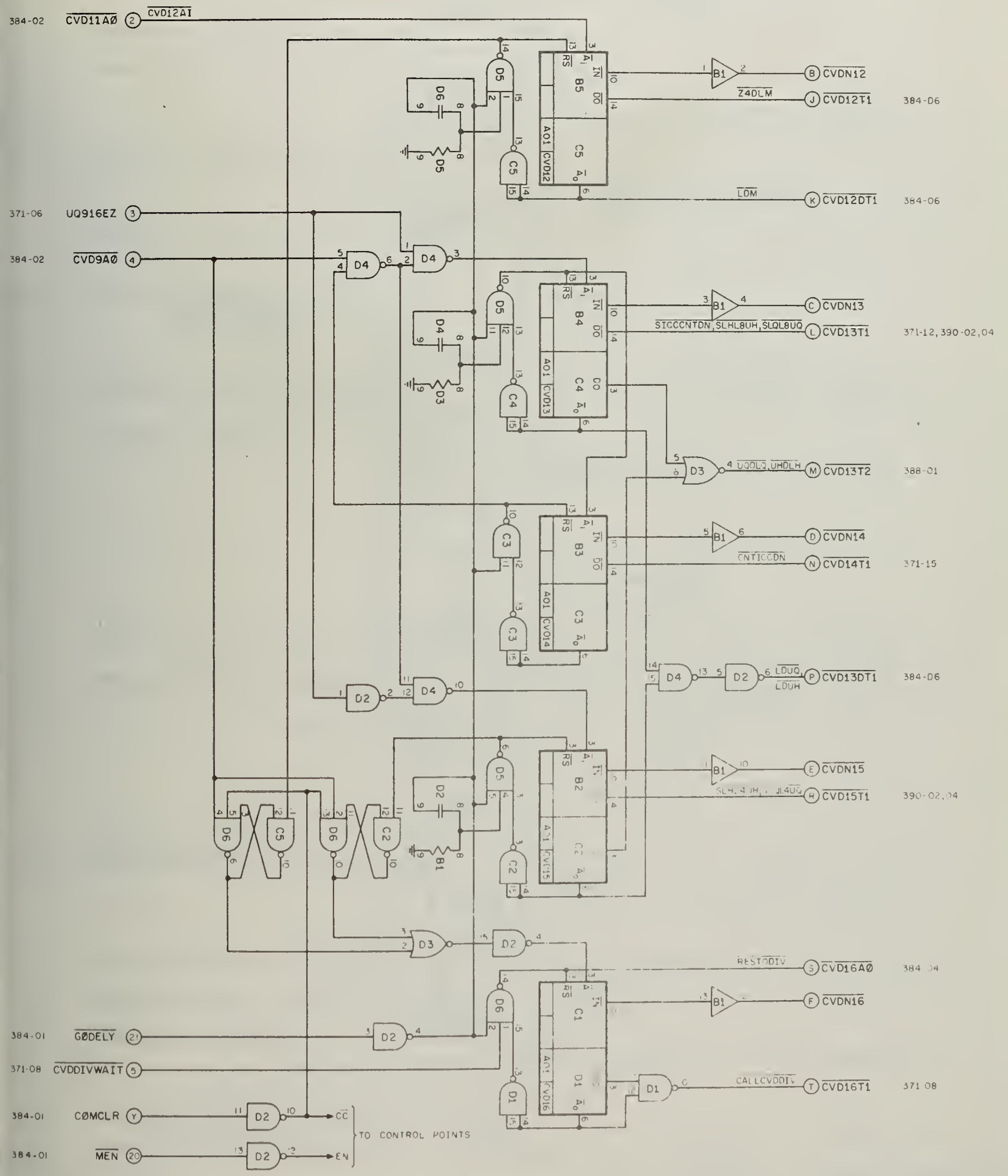
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L. G.	T. G.	12-10-71	
Approved by	Date	Reference	
	1-11-72		

TITLE AU - CVD CONTROL LOGIC
DECIMAL DIGITS GENERATION –
CALCULATE QUOTIENT & REMAINDER
-GETDEC – CAQOREM-

Sheet ____ of ____ Drawing No. AUO-07-384-2



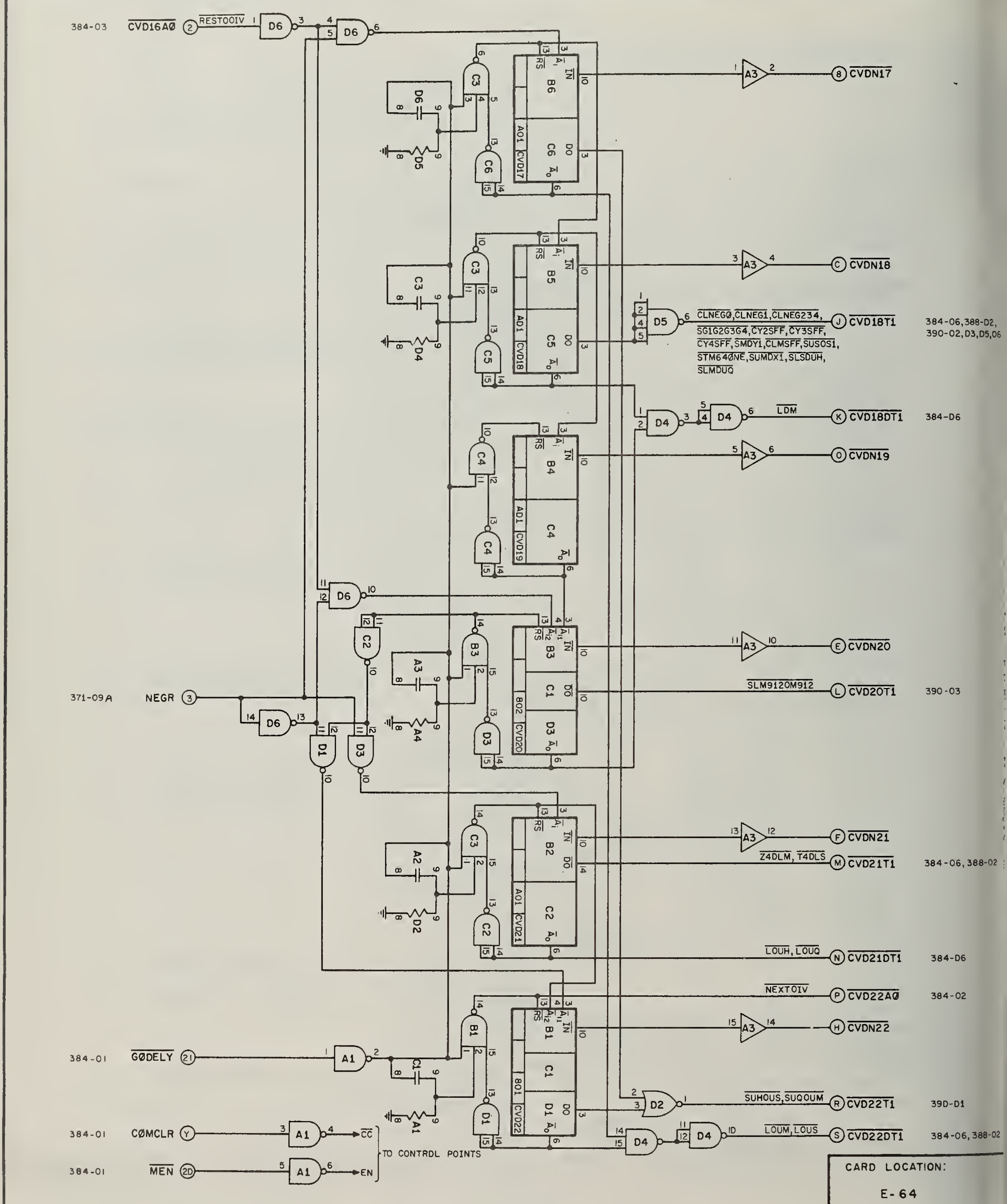
Issue	Change order	Approved by	Date
REVISIONS			

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg
L G	T G	12-7-71	
Approved by	Date	Reference	

TITLE AU - CVD CONTROL LOGIC
DECIMAL DIGITS GENERATION -
CALCULATE QUOTIENT & REMAINDER
- GETDEC - CAQOREM -

Sheet ____ of ____ Drawing No. AUO-07-384-03



Issue	Change order	Approved by	Date
2	NO. 539	P. Krabbe	3-30-72

REVISIONS

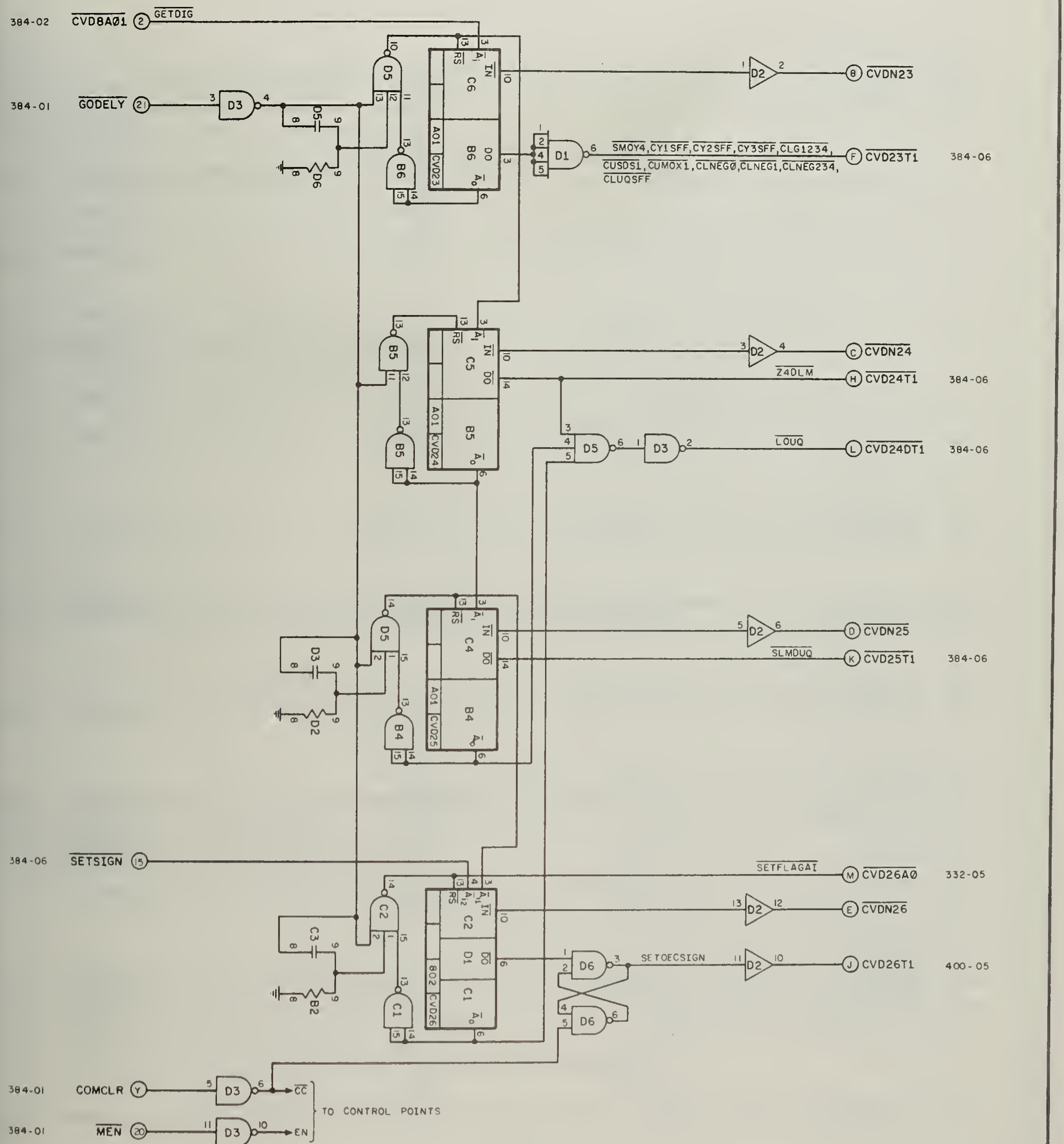
DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For: L. G. | Drawn by: T. G. | Date: 12-22-71 | Supersedes dwg.

Approved by: [signature] | Date: 1-4-72 | Reference

TITLE: AU-CVD CONTROL LOGIC
DECIMAL DIGIT GENERATION—
QUOTIENT DECREMENT
GETDEC-QUODCM

Sheet ____ of ____ Drawing No. AUO-07-384-04



CARD LOCATION:
E-66

Issue	Change order	Approved by	Date

REVISIONS

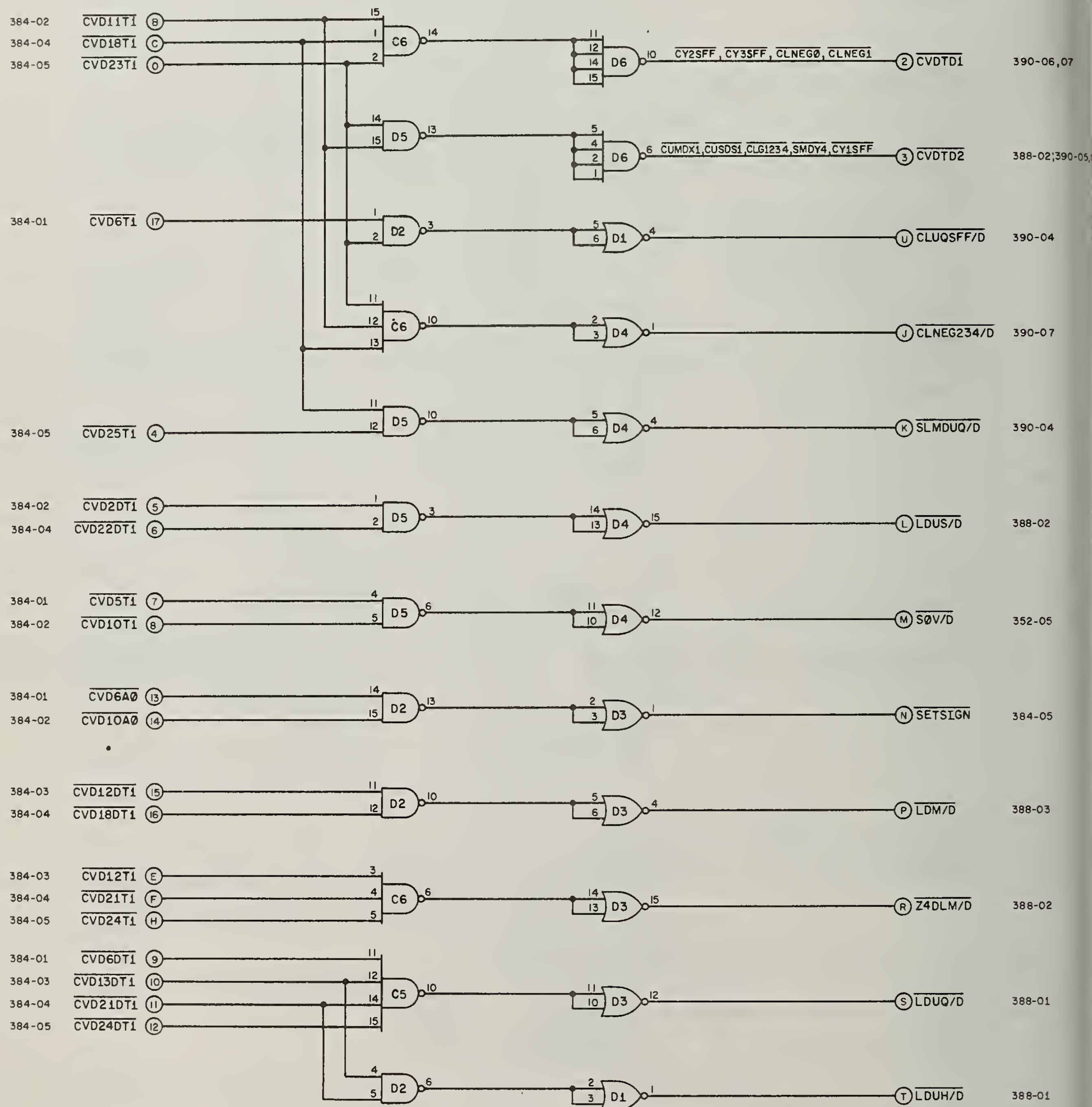
DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L G	T.G	1-21-72	
Approved by	Date	Reference	
[illegible]	1 27 172		

TITLE AU – CVD CONTROL LOGIC
DECIMAL DIGITS TO UQ
GETDIG AND SETSIGN

Sheet ____ of ____

Drawing No. AUO – 07 – 384 – 05



CARD LOCATION:
E-62

Issue	Change order	Approved by	Date

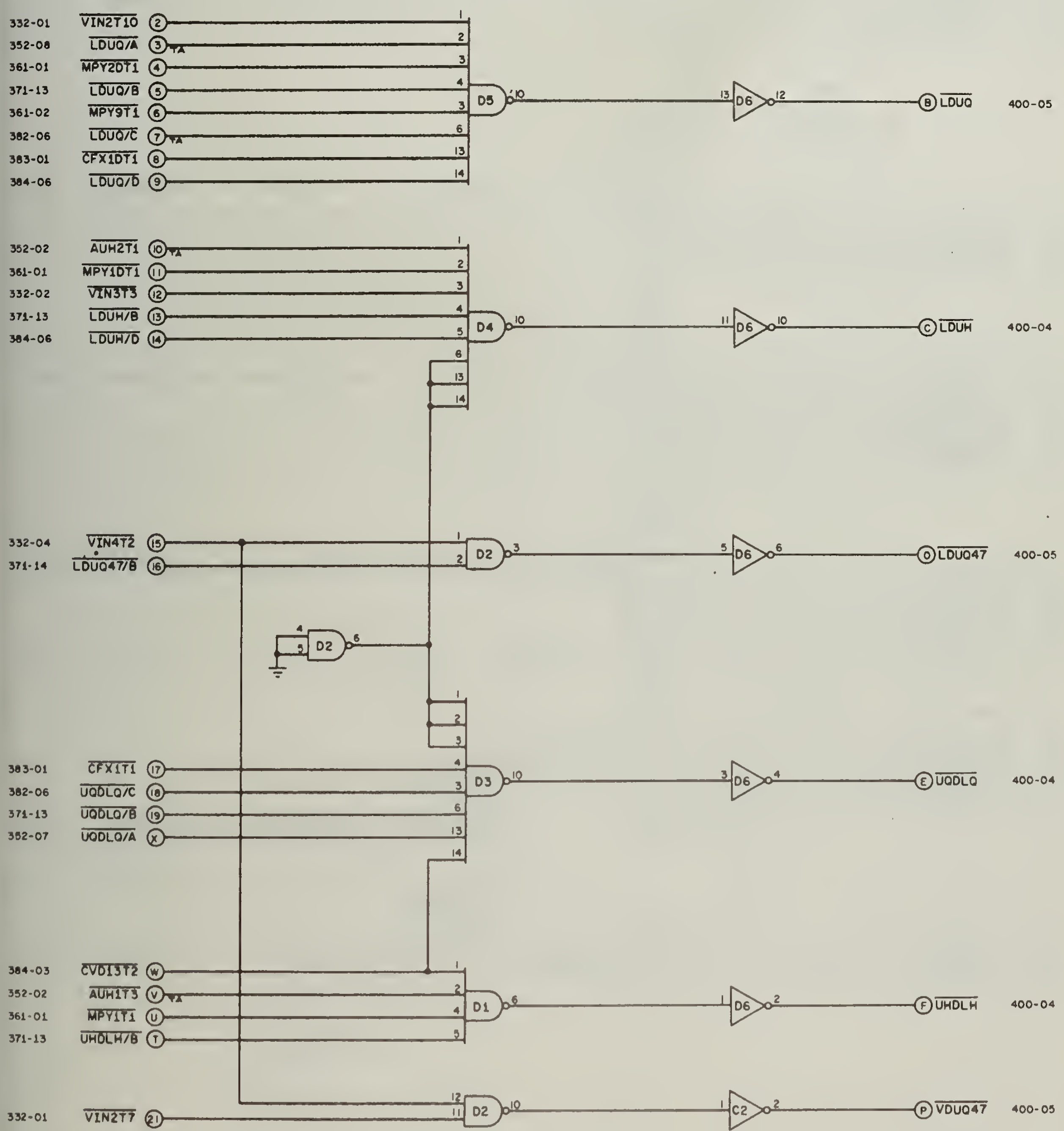
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L.G.	T.G.	11-10-71	
Approved by	**Date**	**Reference**	
[illegible signature]	1-11-72		

TITLE AU - CVD CONTROL LOGIC
TASK SIGNAL COLLECTOR "D"

Sheet ____ of ____ Drawing No. AUO-07-384-06



CARD LOCATION:
H-64

Issue	Change order	Approved by	Date
2	NO. 558	C. Knabbe	4-12-72

REVISIONS

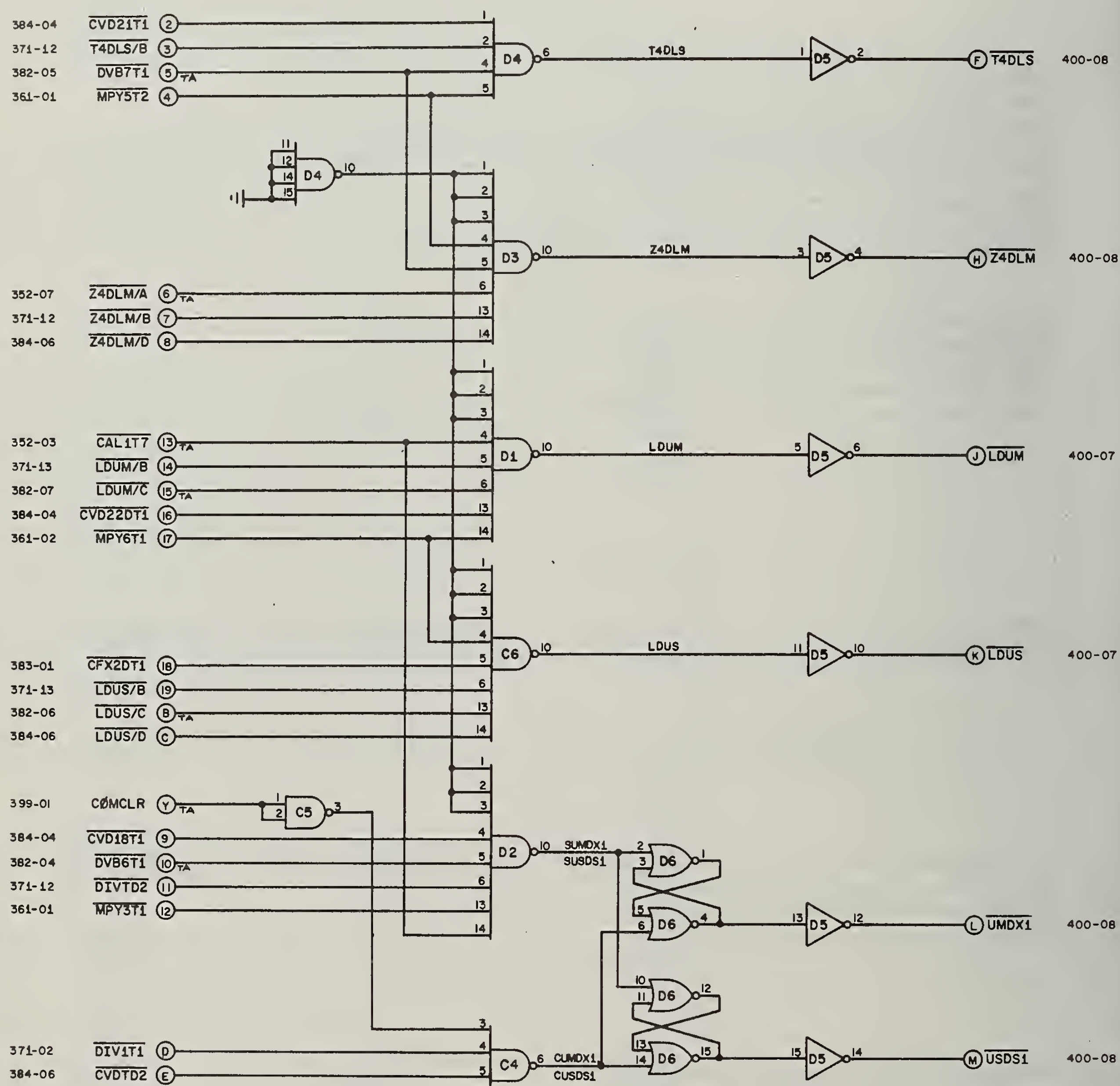
DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L G	T G	11-8-71	
Approved by	Date	Reference	
Gajal	1-11-72		

TITLE AU FINAL LEVEL TASK DRIVERS
TASK DRIVER "F1"

Sheet____of____

Drawing No. AUO-07-388-01



CARD LOCATION: H-62

Issue	Change order	Approved by	Date
3	NO. 709	E N Goyal	3-8-73
2	NO. 554	P. Kraskie	4-12-72

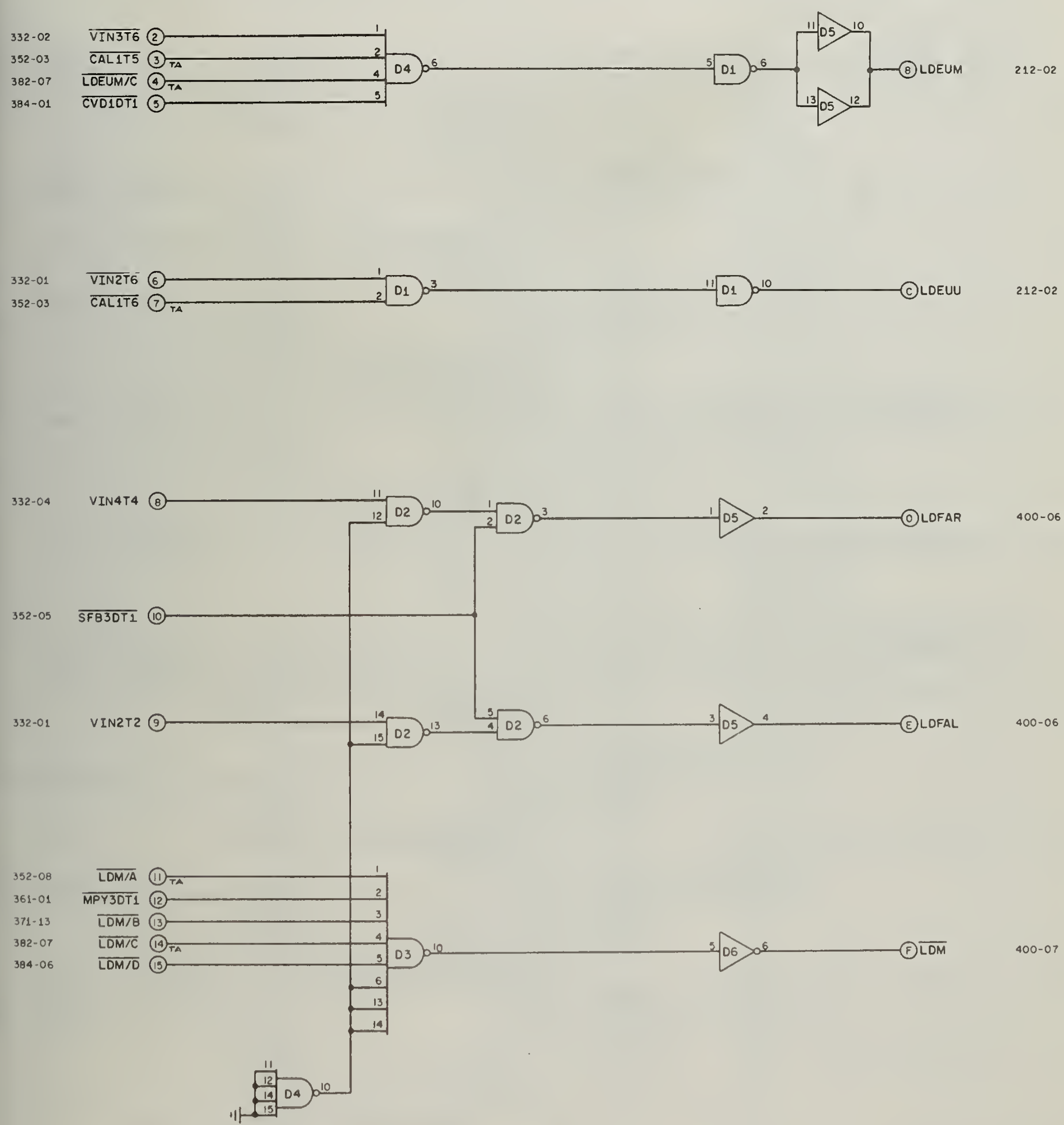
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L. G.	T. G.	11-8-71	
Approved by	Date	Reference	
[illegible]	1-11-72		

TITLE AU FINAL LEVEL TASK DRIVERS
TASK DRIVER "F2"

Sheet ____ of ____ Drawing No. AUO-07-388-0



CARD LOCATION:
H-66

Issue	Change order	Approved by	Date
3	No. 621	P. Kraljic	8-10-72
2	No. 546	P. Kraljic	1-12-72

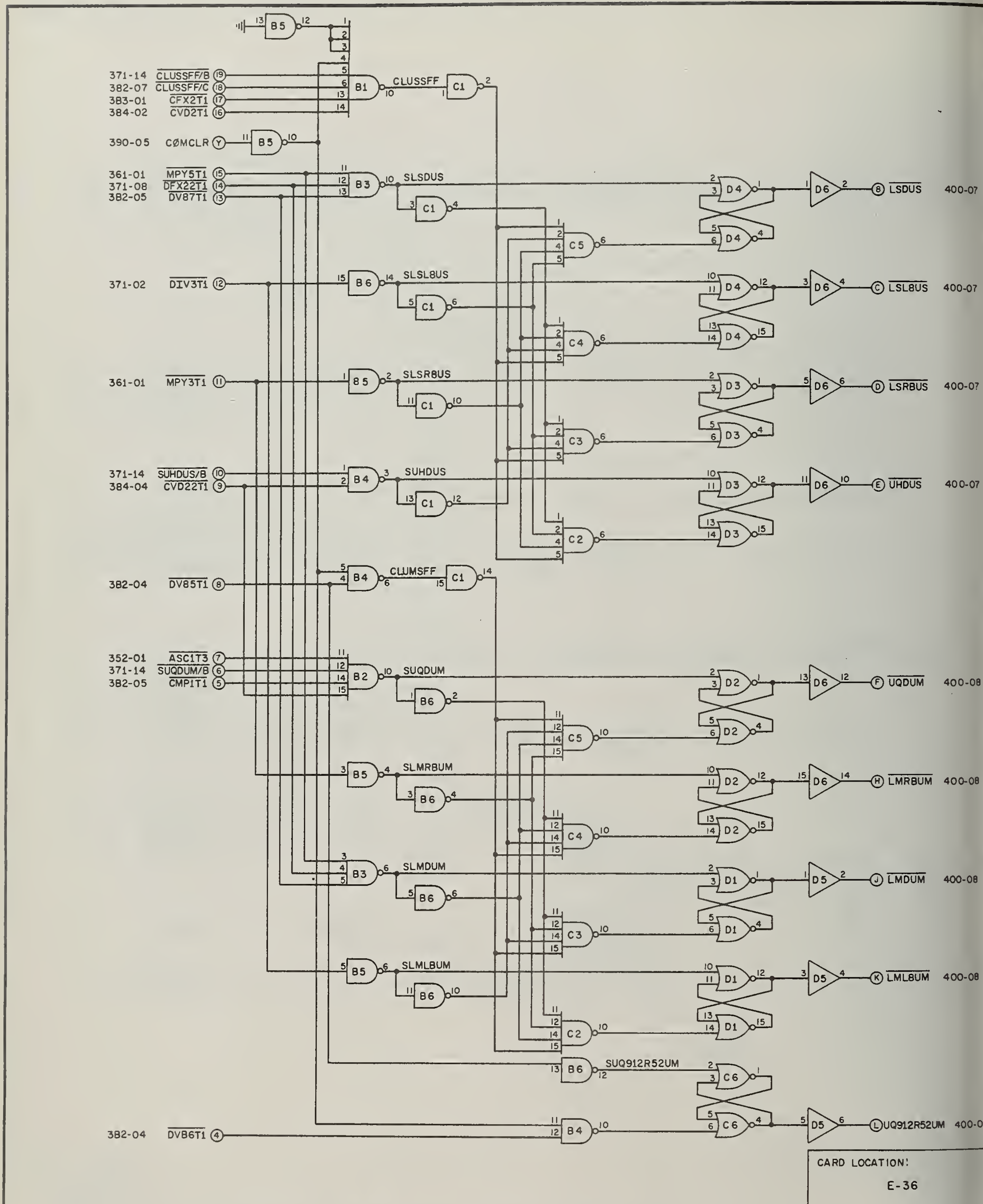
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L G	T G.	11-9-71	
Approved by	Date	Reference	
Goyal	1-11-72		

TITLE AU FINAL LEVEL TASK DRIVERS
TASK DRIVER "F3"

Sheet____ of ____ Drawing No. AUO-07-388-03



CARD LOCATION:
E-36

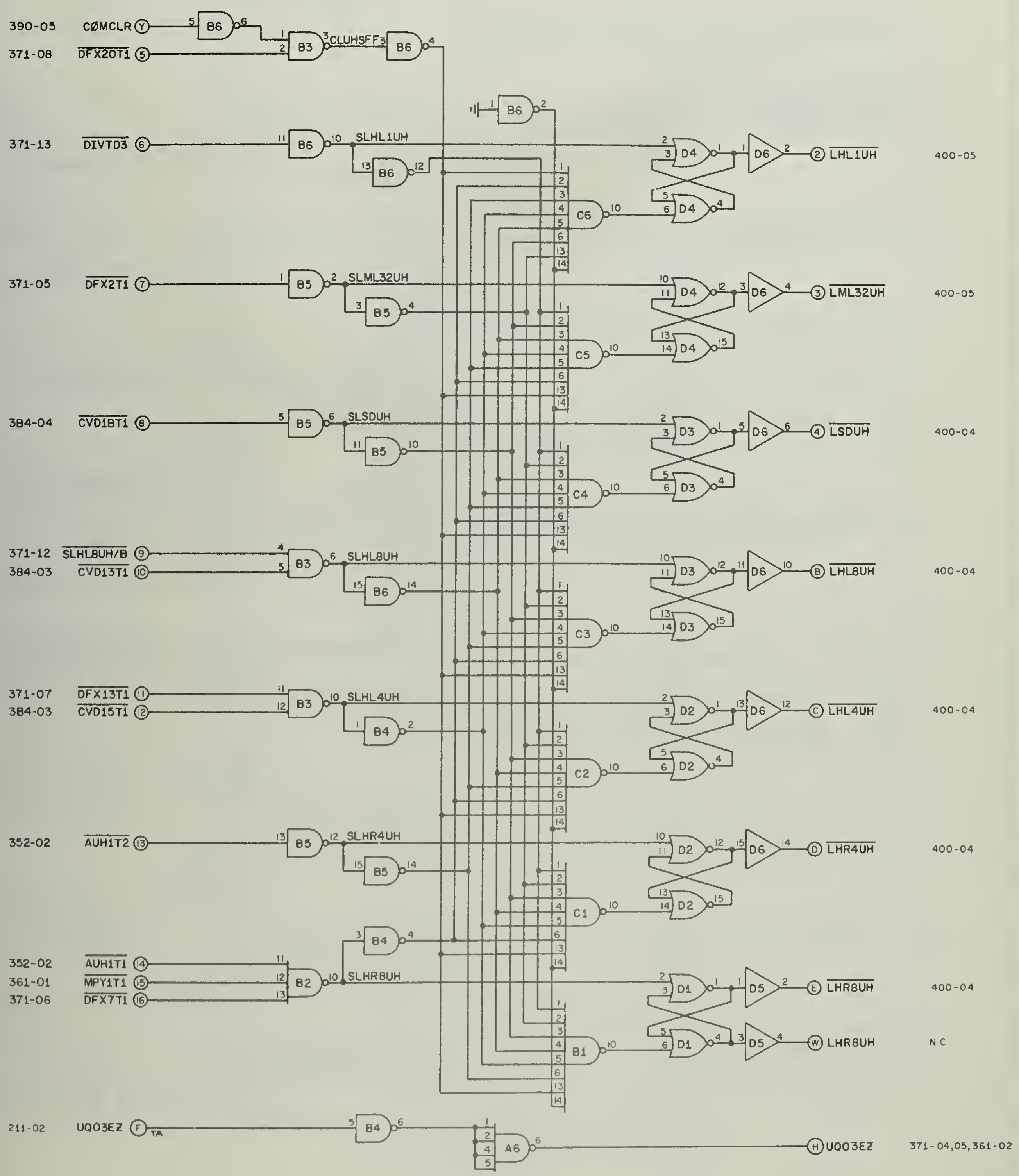
Issue	Change order	Approved by	Date
REVISIONS			

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L. G.	T. G.	7-19-71	
Approved by	Date	Reference	
	1-11-72		

TITLE AU REGISTER SELECTORS
UM, US REGISTER SELECTORS

Sheet ____ of ____ Drawing No. AUO-07-390-1



CARD LOCATION:

E38

Issue	Change order	Approved by	Date
3	NO.604	[illegible]	5-25-72
2	No 580	[illegible]	4-4-72

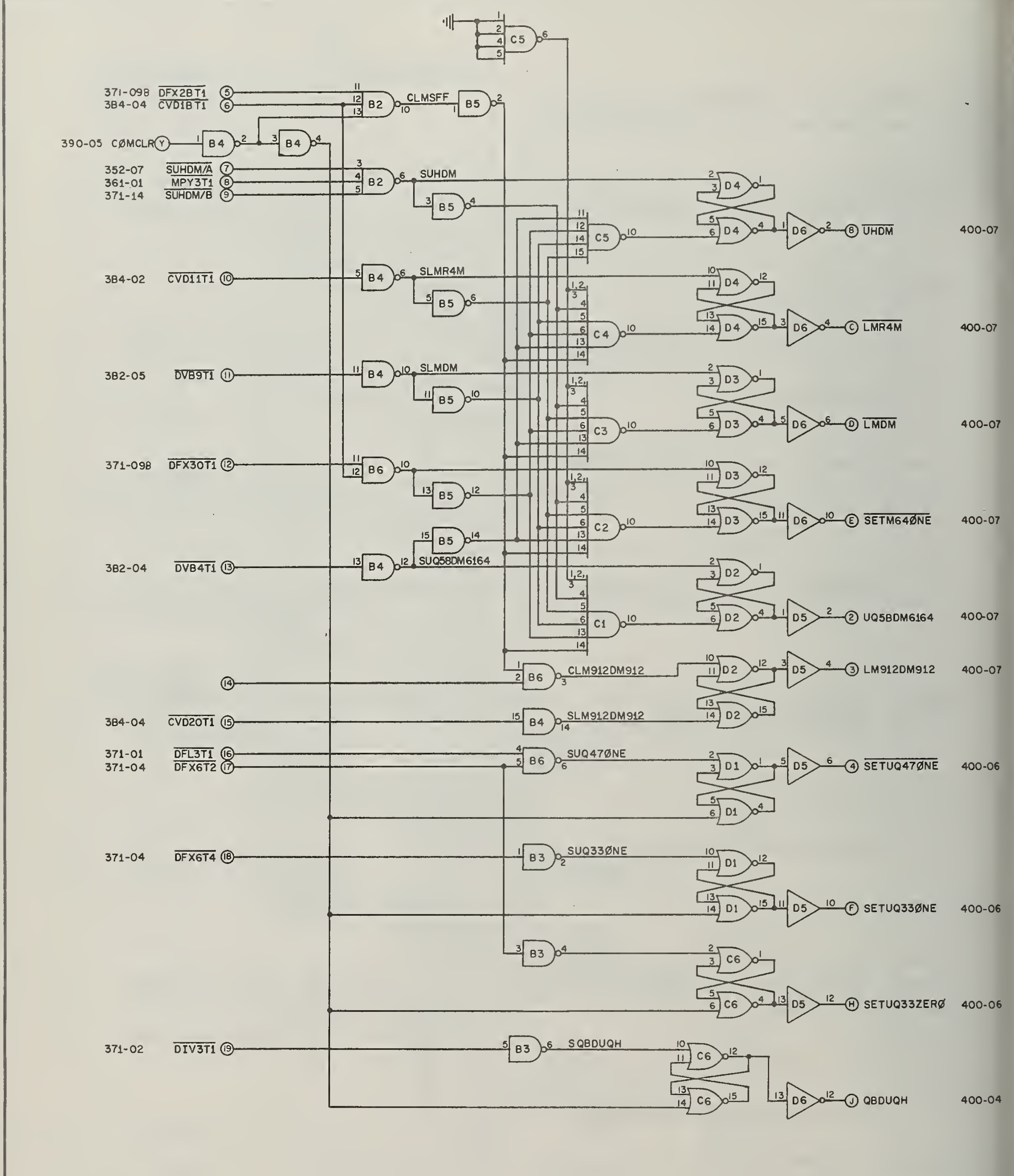
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L G	T G	7-23-71	
Approved by	Date	Reference	
[illegible]	1-41-72		

TITLE AU REGISTER SELECTORS
UH REGISTER SELECTOR

Sheet ____ of ____ Drawing No. AUO-07-390-02



CARD LOCATION:

E-46

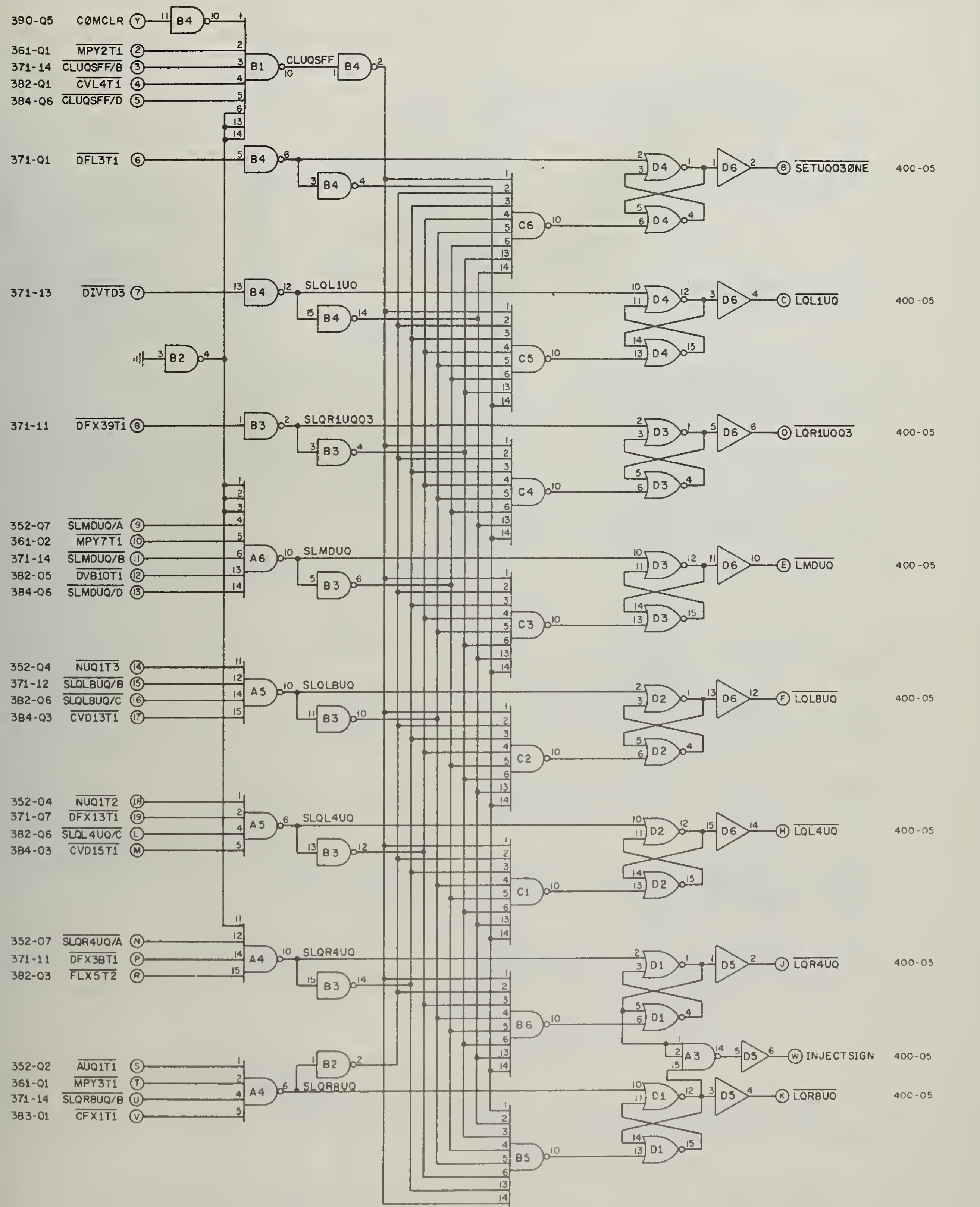
Issue	Change order	Approved by	Date
REVISIONS			

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L.G.	T.G.	7-22-71	
Approved by	Date	Reference	
[illegible signature]	1-11-72		

TITLE AU REGISTER SELECTORS
M REGISTER SELECTOR,
SETUQ33ØNE, SETUQ33ZERØ, SETUQ47ØNE, QBDUQH

Sheet ____ of ____ Drawing No. AUO-07-390-03



CARD POSITION:

E-40

Issue	Change order	Approved by	Date
3	No 654	P. Kraft	1-30-73
2	No 581	P. Kraft	1-4-72

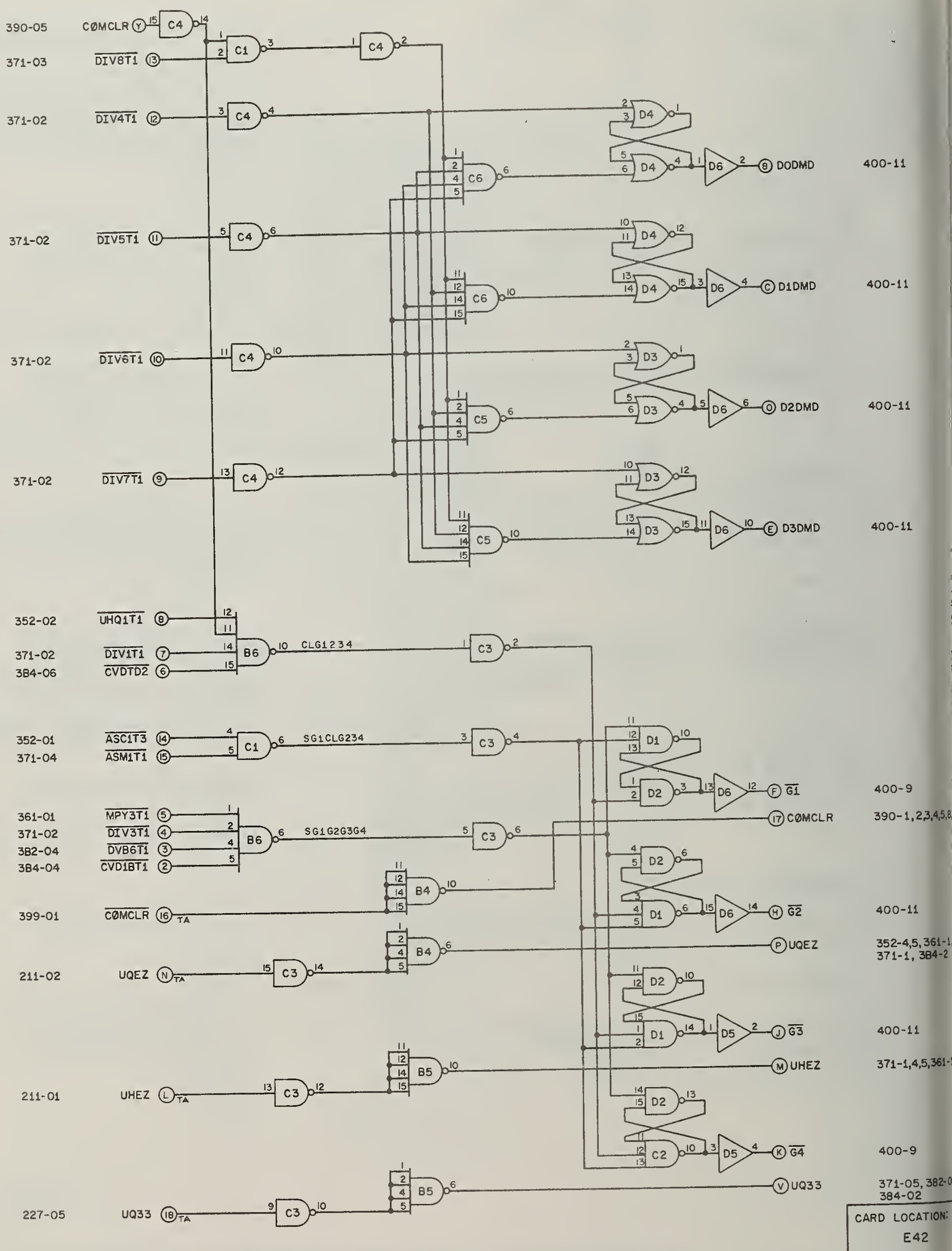
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For: L.G. | Drawn by: T G | Date: 7-23-71 | Supersedes dwg.:
Approved by: | Date: 1-11-72 | Reference:

TITLE AU REGISTER SELECTORS
UQ REGISTER SELECTOR

Sheet ____ of ____ Drawing No. AUO-07-390-04



CARD LOCATION: E42

Issue	Change order	Approved by	Date
2	NO. 605	P. Kralle	5-15-72

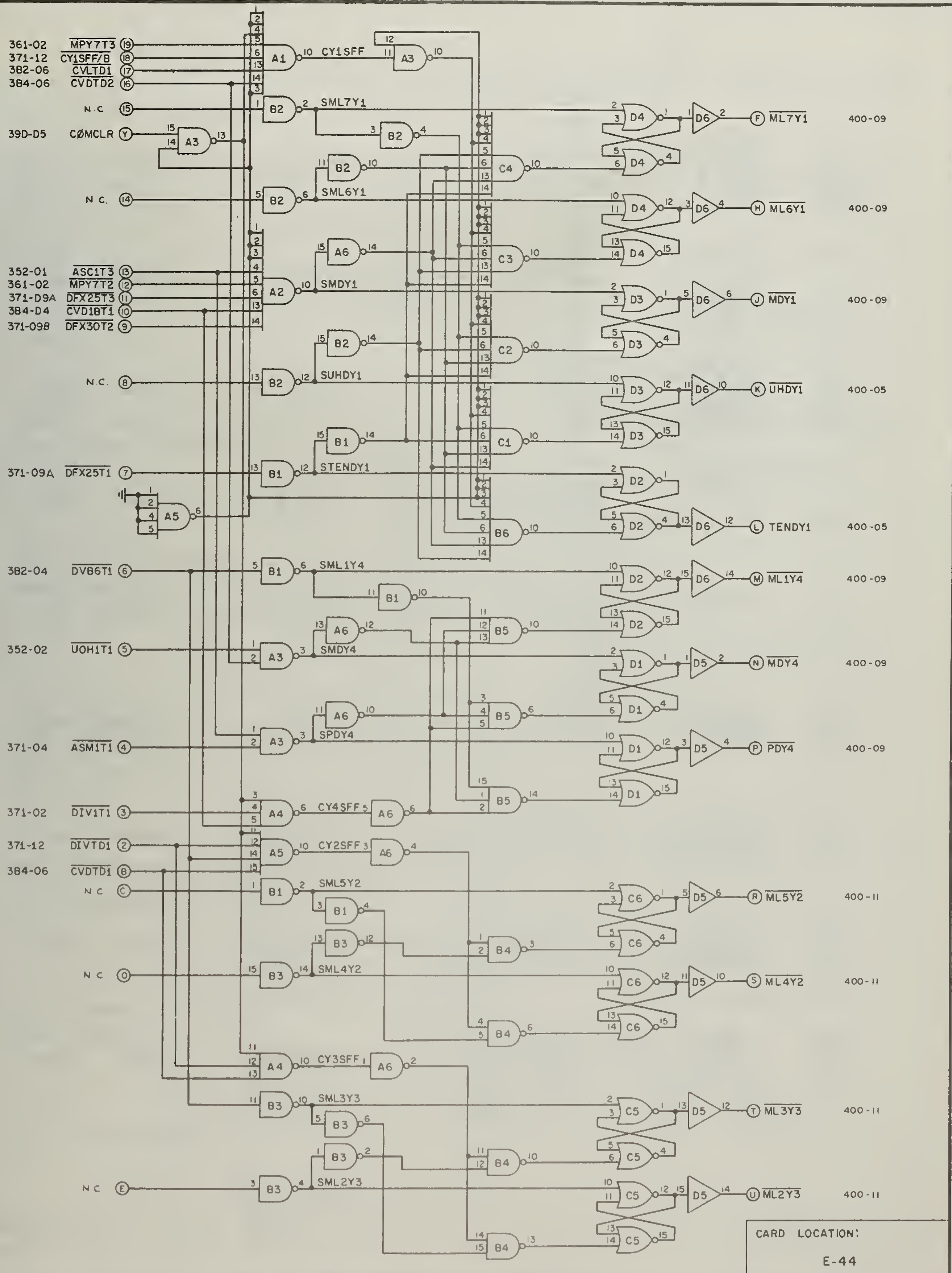
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L. G.	T. G.	8-1-71	
Approved by	Date	Reference	
	1-41-72		

TITLE AU REGISTER SELECTORS
MODEL DIVISION DIVISOR SELECTORS,
SDS GATE SIGNAL Gi SELECTORS

Sheet ___ of ___ Drawing No. AUO-07-390-05



CARD LOCATION:

E-44

2	No 552	P Kralike	4.4.72
Issue	Change order	Approved by	Date

REVISIONS

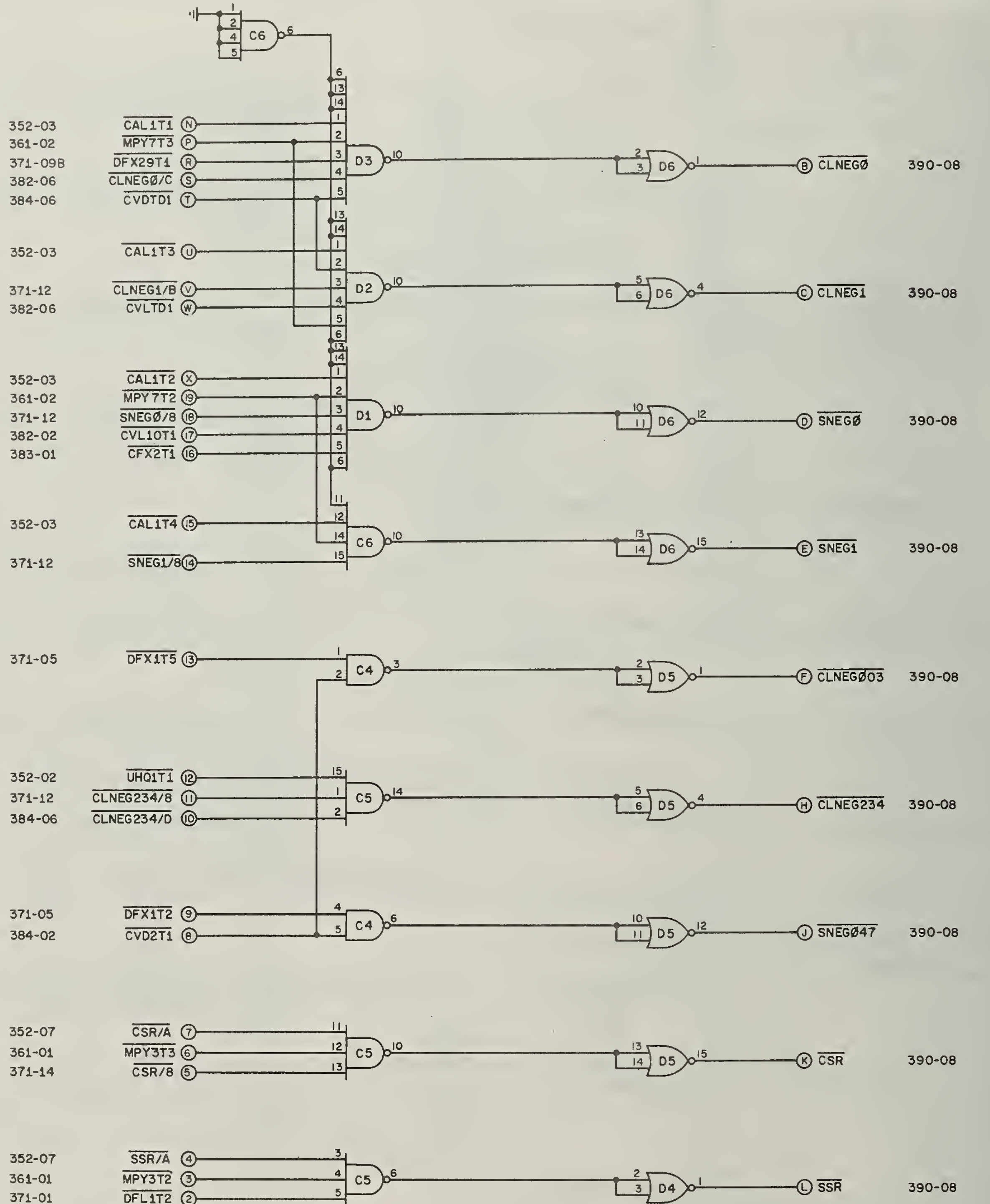
DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg
L. G.	T G	7-21-71	
Approved by	Date	Reference	
	11 72		

TITLE AU REGISTER SELECTORS
M SHIFT ARRAY SELECTORS

Sheet ____ of ____

Drawing No. AUO-07-390-06



CARD LOCATION:

E-34

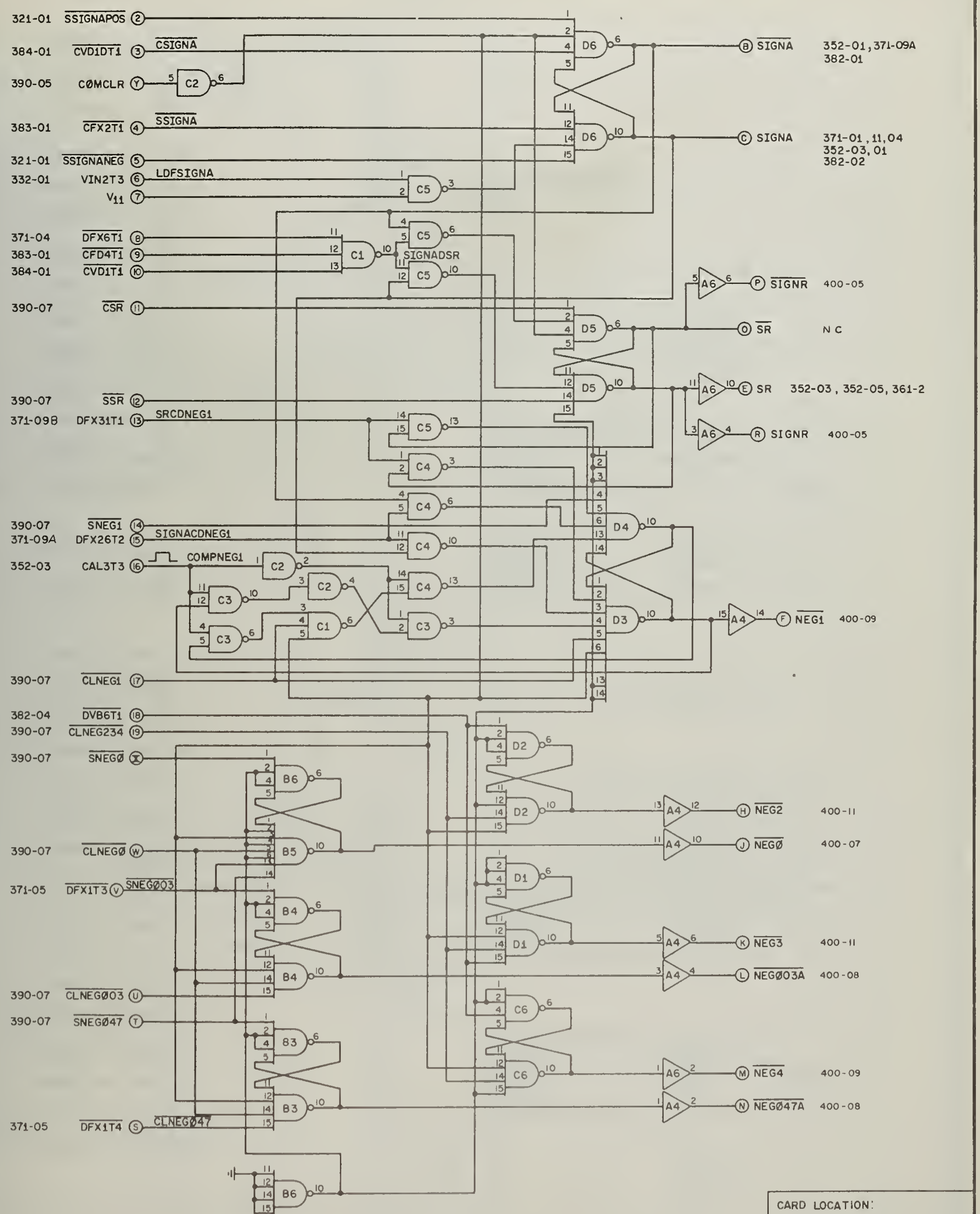
Issue	Change order	Approved by	Date
REVISIONS			

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L. G.	TG	7-28-71	
Approved by	**Date**	**Reference**	
[signature]	1-11-72		

TITLE AU REGISTER SELECTORS
TASK SIGNAL COLLECTOR

Sheet ____ of ____ Drawing No. AUO-07-390-07



CARD LOCATION:

E-32

Issue	Change order	Approved by	Date
3	No 622	[illegible]	8-10-72
2	No 553	[illegible]	4-4-72

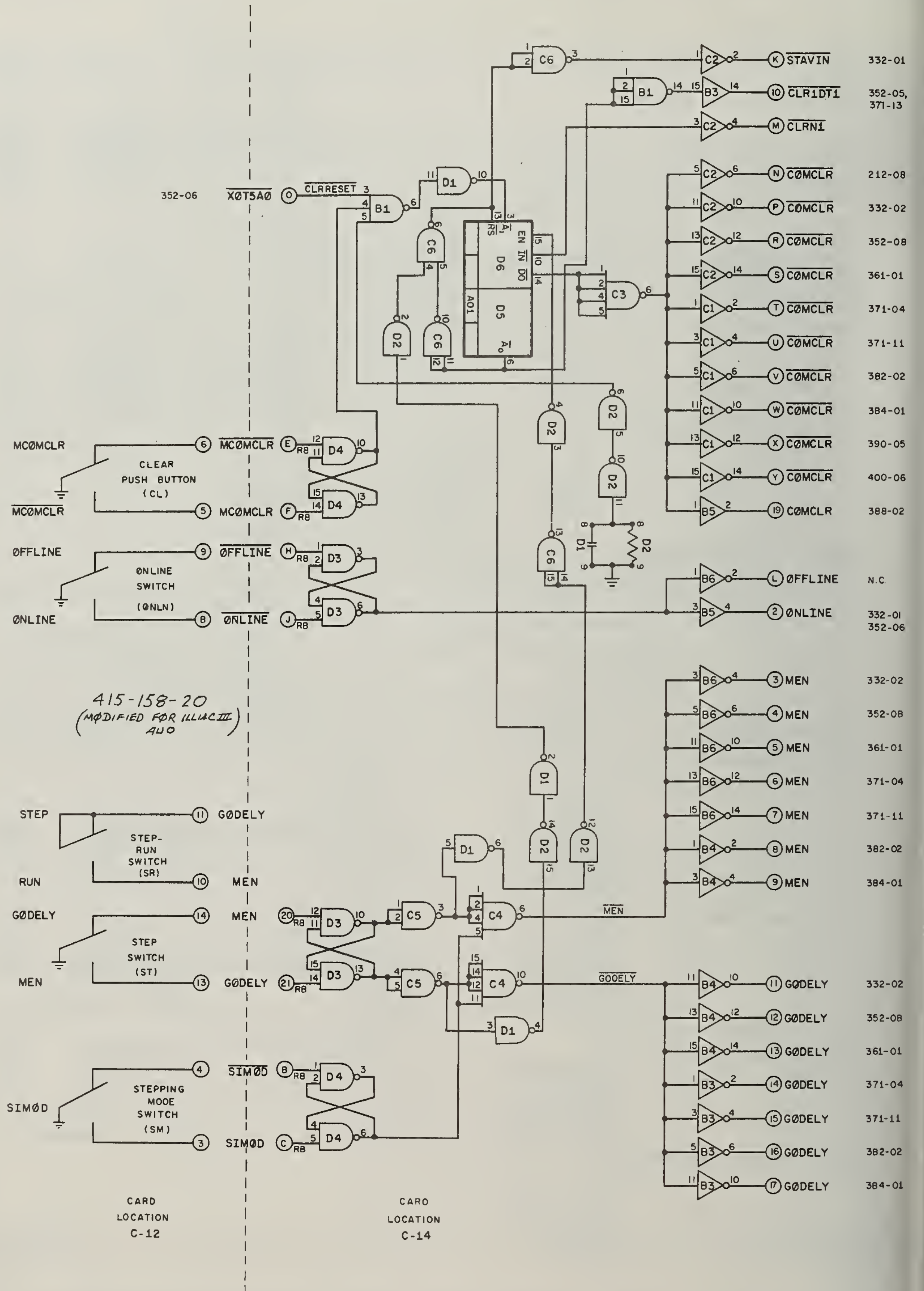
REVISIONS

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
L. G.	T. G.	7-27-71	
Approved by	**Date**	**Reference**	
[illegible]	1-41-72		

TITLE AU REGISTER SELECTORS
SDS NEGi SIGNAL SELECTORS, SR, SIGNA

Sheet ____ of ____ Drawing No. AUO-07-390-08



DEPARTMENT of COMPUTER SCIENCE
University of Illinois

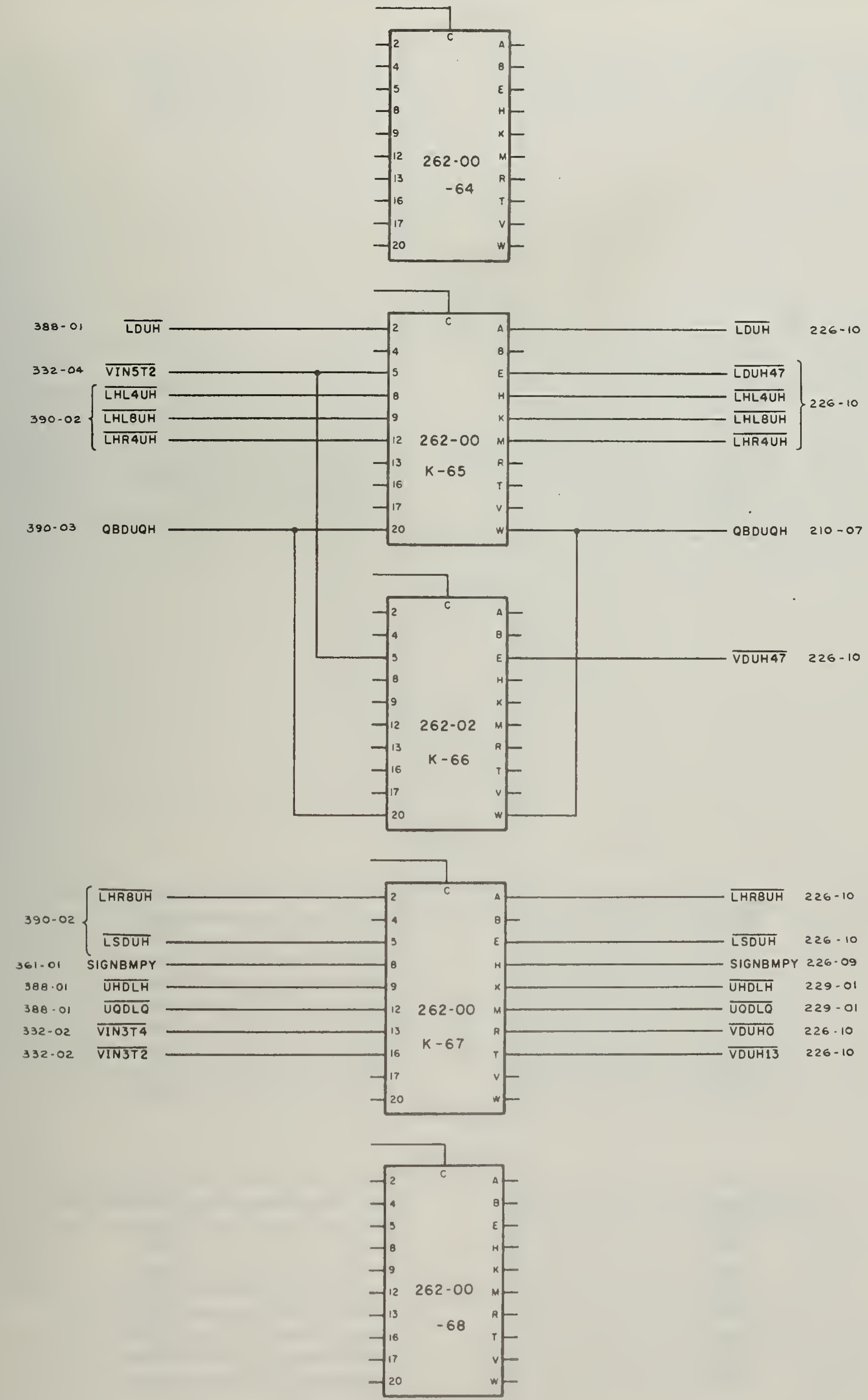
For	Drawn by	Date	Supersedes dwg.
PK	T.G.	4-17-72	
Approved by	Date	Reference	
Paul Kralike	4-20-72		

TITLE AU INITIALIZE CONTROL LOGIC
CREST-CLEAR RESET, POWER TURN-ON & SINGLE STEP MODE LOGIC

Sheet ____ of ____ Drawing No. AUO-07-399-

Issue	Change order	Approved by	Date

REVISIONS



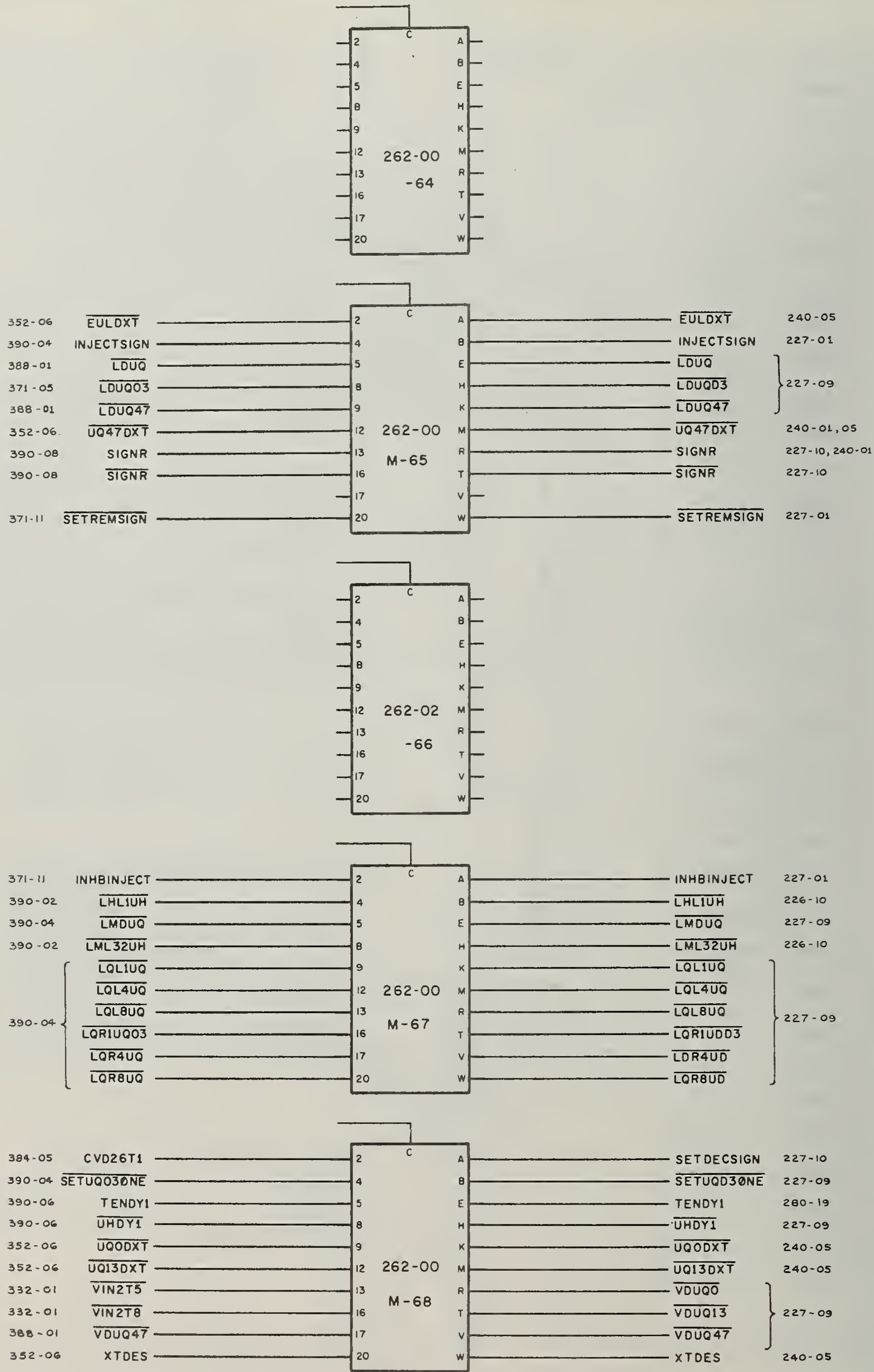
Issue	Change order	Approved by	Date
REVISIONS			

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For: P K | Drawn by: S Z | Date: 3-7-72 | Supersedes dwg.:
Approved by: P. Krabbe | Date: 3-10-72 | Reference:

TITLE: AU DRIVER INTERFACE
CONTROL → PROCESSING HARDWARE

Sheet ____ of ____ | Drawing No. AUO-07-400-04



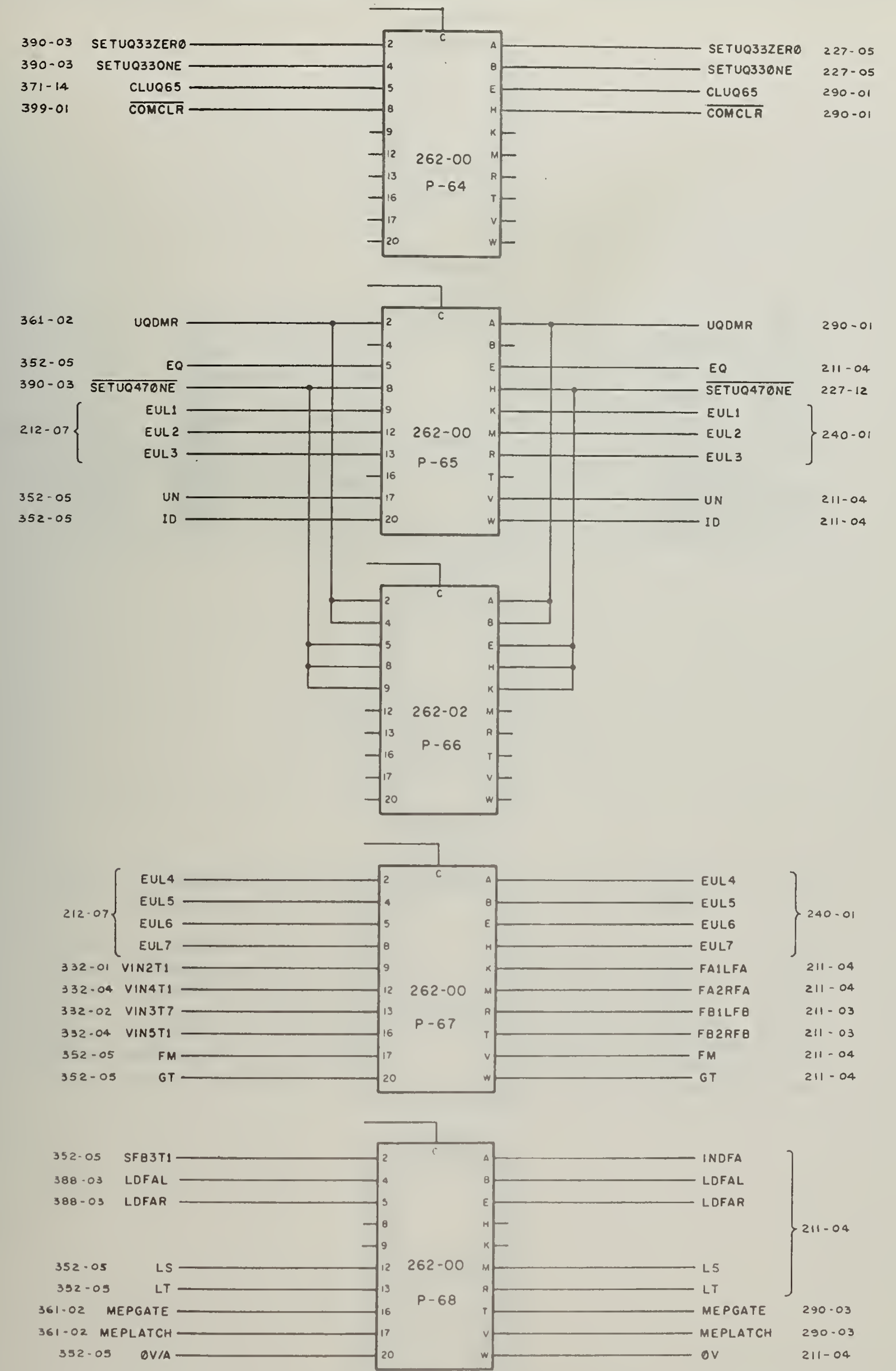
Issue	Change order	Approved by	Date
REVISIONS			

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
P K	S Z	3-7-72	
Approved by: P Krabbe	Date: 3-10-72	Reference	

TITLE
AU DRIVER INTERFACE
CONTROL → PROCESSING HARDWARE

Sheet ____ of ____ Drawing No. AUO-07-400-5



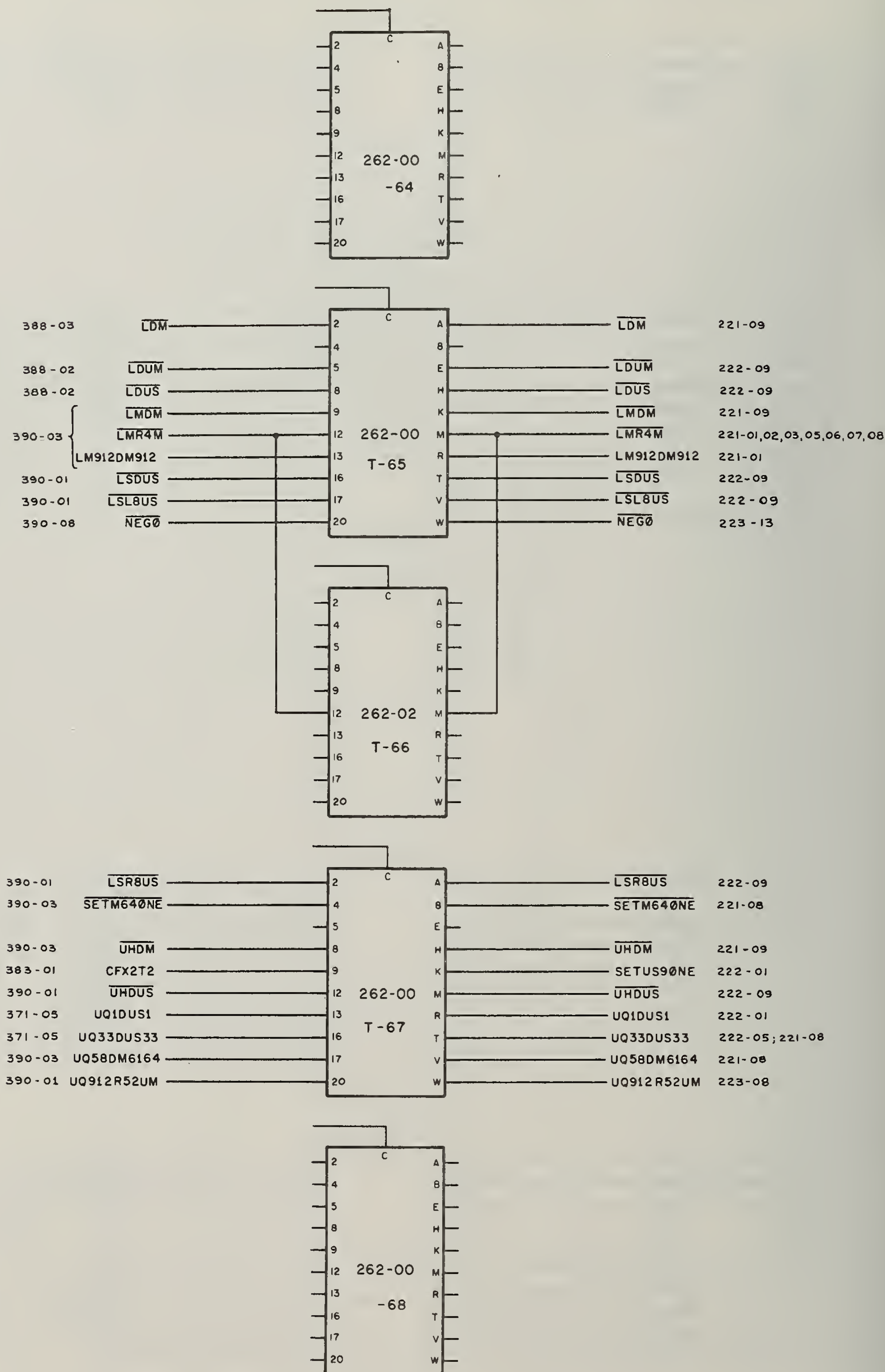
Issue	Change order	Approved by	Date
REVISIONS			

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
P K	S Z	3-7 72	
Approved by	**Date**	**Reference**	
P Kralla	3-10-72		

TITLE AU DRIVER INTERFACE
CONTROL → PROCESSING HARDWARE

Sheet ____ of ____ Drawing No. AUO-07-400-06



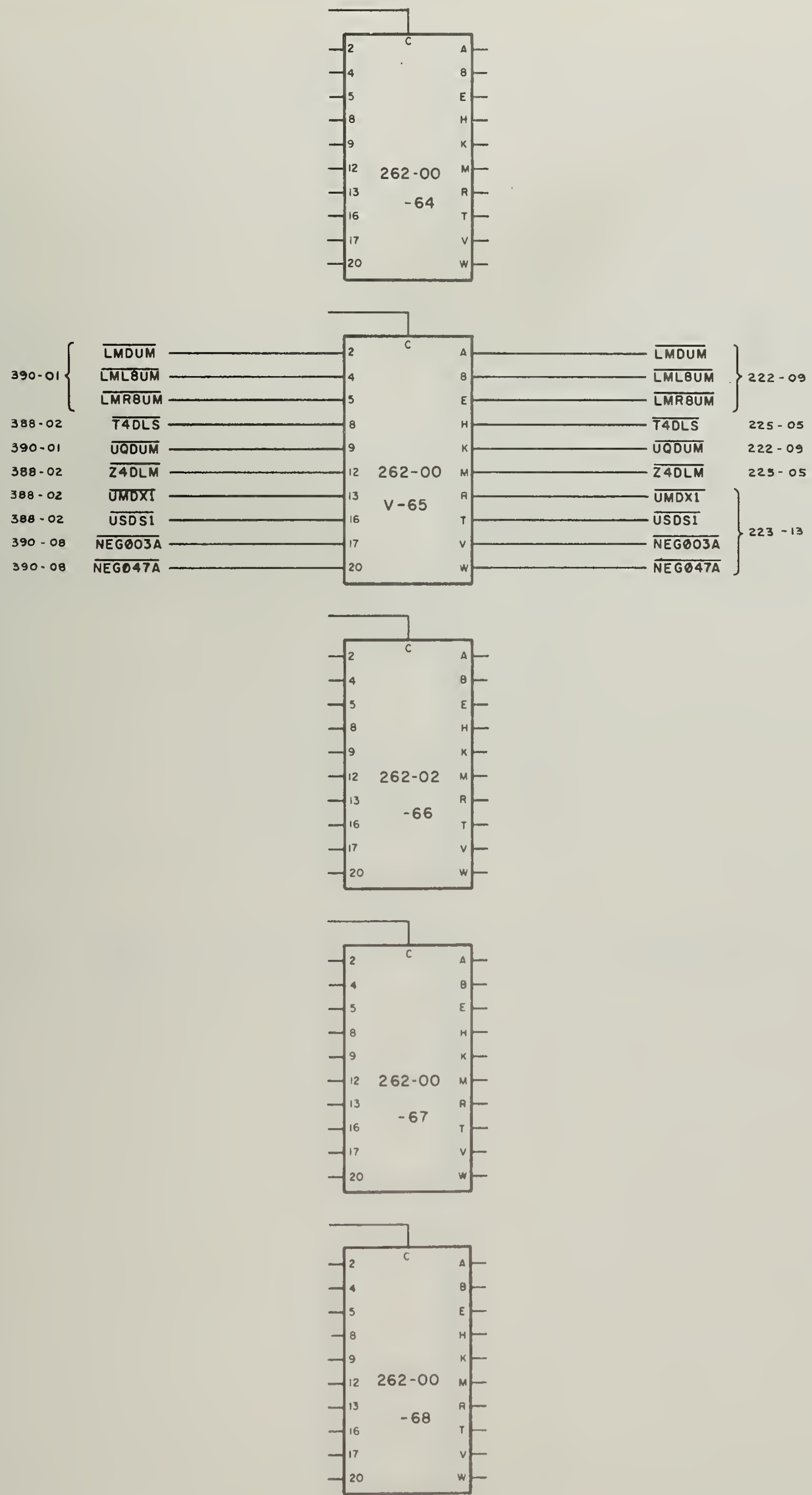
Issue	Change order	Approved by	Date
REVISIONS			

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
P K	S Z	3-7-72	
Approved by	Date	Reference	
P Krabbe	3-10-72		

TITLE AU DRIVER INTERFACE
CONTROL → PROCESSING HARDWARE

Sheet ____ of ____ Drawing No. AUO-07-400-0



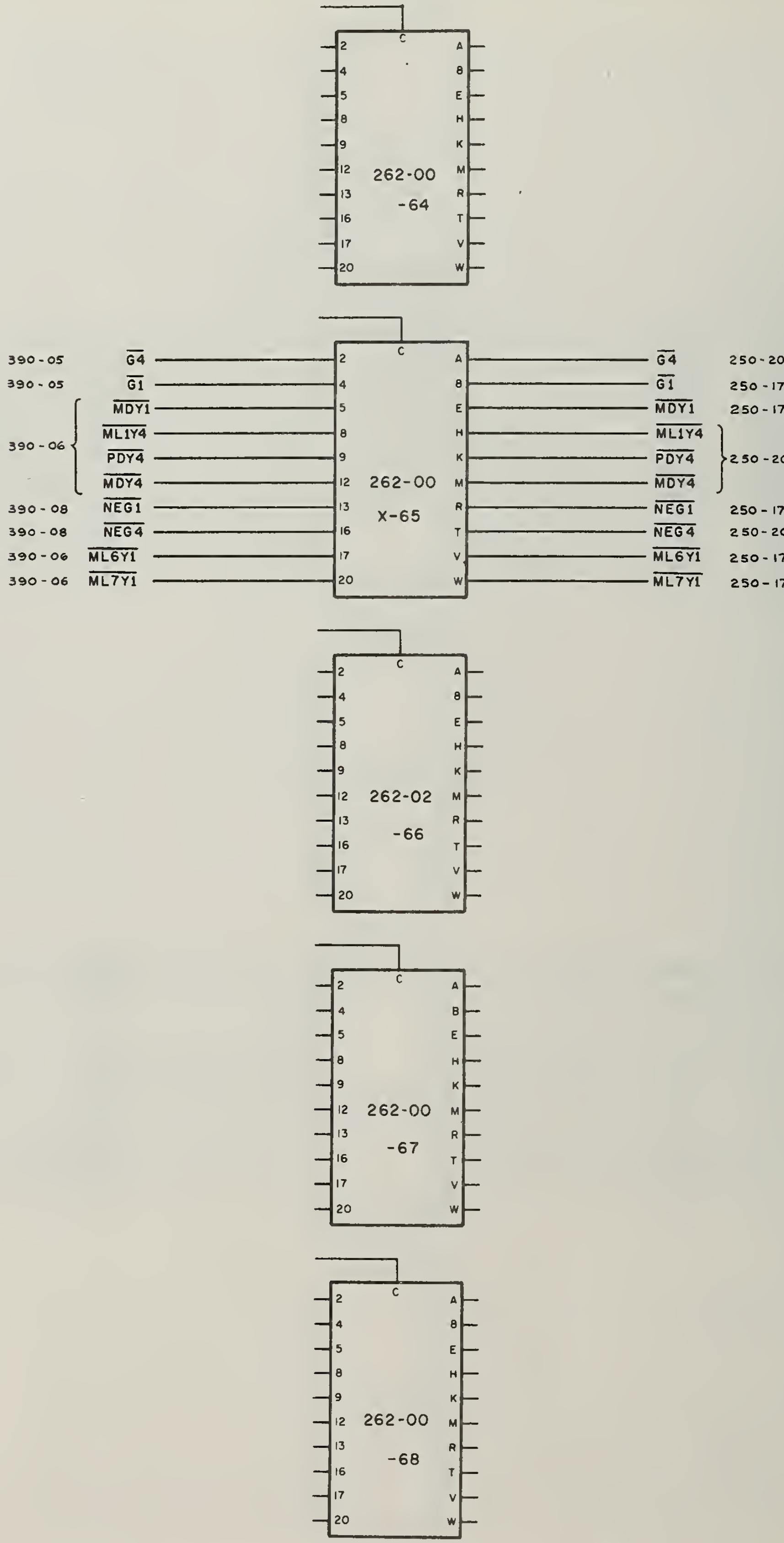
Issue	Change order	Approved by	Date
REVISIONS			

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
P K	S Z	3-7-72	
Approved by	Date	Reference	
P Kraska	3-13 72		

TITLE AU DRIVER INTERFACE
CONTROL → PROCESSING HARDWARE

Sheet ____ of ____ Drawing No. AUO-07-400-08



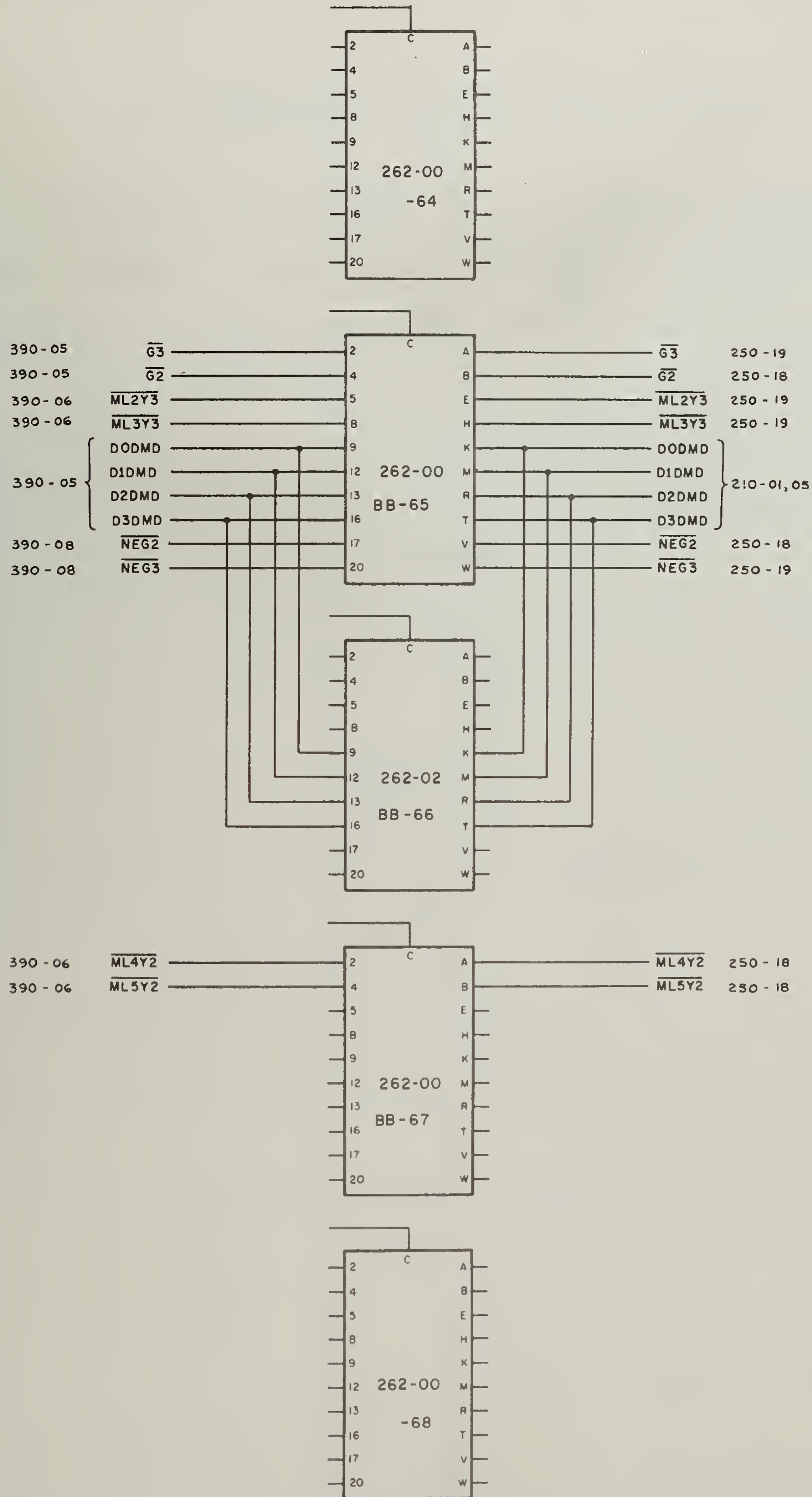
Issue	Change order	Approved by	Date
REVISIONS			

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
P K	S Z	3-7-72	
Approved by	Date	Reference	
P Krabbe	3-13-72		

TITLE AU DRIVER INTERFACE
CONTROL → PROCESSING HARDWARE

Sheet ____ of ____ Drawing No. AU0-07-400-09



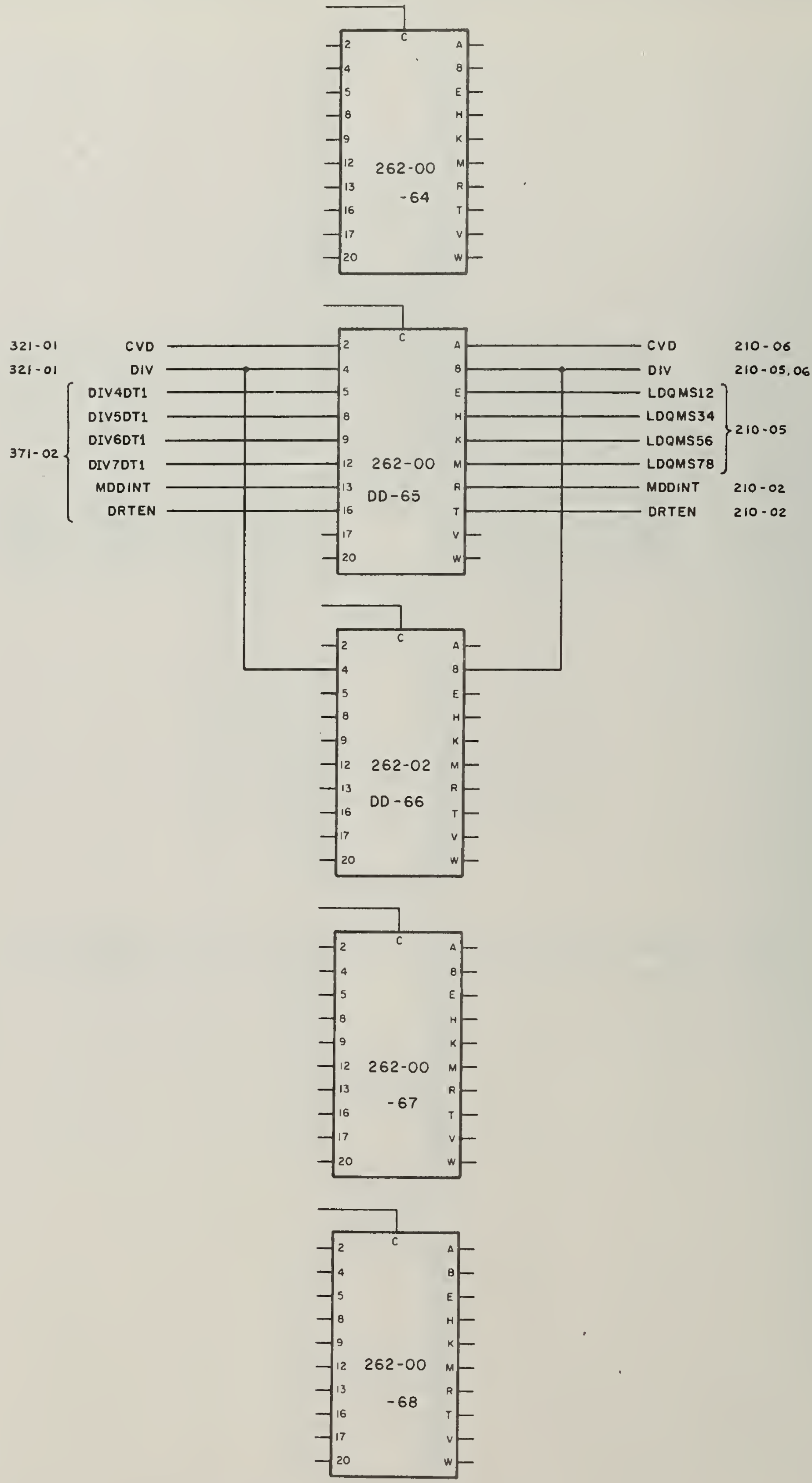
DEPARTMENT of COMPUTER SCIENCE
University of Illinois

Issue	Change order	Approved by	Date
REVISIONS			

For	Drawn by	Date	Supersedes dwg.
P K	S Z	3-7-72	
Approved by	Date	Reference	
P Kr...	3-13-72		

TITLE AU DRIVER INTERFACE
CONTROL → PROCESSING HARDWARE

Sheet ___ of ___ Drawing No. AUO-07-400-11



Issue	Change order	Approved by	Date
REVISIONS			

DEPARTMENT of COMPUTER SCIENCE
University of Illinois

For	Drawn by	Date	Supersedes dwg.
P K	S Z	3-7-72	
Approved by	Date	Reference	
P. Kraska	3-13-72		

TITLE AU DRIVER INTERFACE
CONTROL → PROCESSING HARDWARE

Sheet ____ of ____ Drawing No. AU0-07-400-12

6.0 CONTROL SIGNAL NAMES AND THEIR FUNCTIONAL DESCRIPTION

This section gives a listing of Control Signals which appear on the Control logic drawings. These signals together with their functional description are printed in an alphabetical order. PL/1 description of many of these signals is also available in Volume 1 of Illiac III Arithmetic Unit manual (DCS Report No. 366). It is necessary for the reader to be conversant with the static description of the AU (date paths, registers' structure etc.) and strategy of control point technique for implementing the control sequences to fully understand the functional description of the signal names.

SIGNAL NAME	SIGNAL FUNCTIONAL DESCRIPTION
ADD	ARITHMETIC ORDER ADD
AEXPGT	A'S EXPONENT IS GREATER
ALNUHWAIT	SUBROUTINE ALIGN UH WAIT
ALNUQWAIT	SUBROUTINE ALIGN UQ WAIT
ASC	ARITHMETIC ORDER ADD OR SUBTRACT OR COMPARE
ASIMWAIT	SUBROUTINE ASSIMILATE WAIT
AUBUSY	AU IS BUSY
BEXASC	BEGIN EXECUTION OF A CONTROL SEQUENCE FOR ASC
BEXCVDFIX	BEGIN EXECTUION OF CONTROL SEQUENCE FOR CVD, FIXED POINT OPERAND
BEXCVDFLT	BEGIN EXECUTION OF CONTROL SEQUENCE FOR CVD, FLOATING POINT OPERAND
BEXCVFDEC	BEGIN EXECUTION OF CONTROL SEQUENCE FOR CVF, DECIMAL OPERAND
BEXCVFIX	BEGIN EXECUTION OF CONTROL SEQUENCE FOR CVF, FIXED POINT OPERAND
BEXCVLDEC	BEGIN EXECUTION OF CONTROL SEQUENCE FOR CVL, DECIMAL OPERAND
BEXCVLFLT	BEGIN EXECUTION OF CONTROL SEQUENCE FOR CVL, FLOATING POINT OPERAND
BEXDIVFIX	BEGIN EXECUTION OF CONTROL SEQUENCE FOR DIV, FIXED POINT OPERANDS
BEXDIVFLT	BEGIN EXECUTION OF CONTROL SEQUENCE FOR DIV, FLOATING POINT OPERANDS
BEXMPYFIX	BEGIN EXECUTION OF CONTROL SEQUENCE FOR MPY, FIXED POINT OPERANDS
BEXMPYFLT	BEGIN EXECUTION OF CONTROL SEQUENCE FOR MPY, FLOATING POINT OPERANDS
BEXPOLY	BEGIN EXECUTION OF CONTROL SEQUENCE FOR POLY.
BR	BOGUS RESULT
CALLALNUH	CALL SUBROUTINE ALIGN UH(ALNUH)
CALLALNUQ	CALL SUBROUTINE ALIGN UQ(ALNUQ)

SIGNAL NAME	SIGNAL FUNCTIONAL DESCRIPTION
CALLASIM	CALL SUBROUTINE ASSIMILATE (ASIM)
CALLCOMPL	CALL SUBROUTINE COMPLEMENT (COMPL)
CALLCVDDIV	CALL SUBROUTINE CVDDIV
CALLDIVIDE	CALL SUBROUTINE DIVIDE
CALLDVB	CALL SUBROUTINE DVB
CALLFLNOFX	CALL SUBROUTINE FL-NORM-FIX
CALLNORMUQ	CALL SUBROUTINE NORMALISE UQ (NORMUQ)
CALLUHTUQ	CALL SUBROUTINE UHTUQ
CAL1AI	CONTROL POINT CAL1 ADVANCE-IN
CBYONE	CLEAR FLIP-FLOP BYONE
CEUCTRDIR	CLEAR FLIP-FLOP EUCTRDIR (EUC COUNTER DIRECTION)
CEUDIFF	CLEAR FLIP-FLOP EUDIFF (EXPONENT DIFFERENCE)
CLDDMDSFF	CLEAR DDMD (DIVIDEND DIRECT TO MODEL DIVISION) SELECTOR FLIP-FLOPS
CLDVBWAIT	CLEAR FLIP-FLOP DVBWAIT
CLEUMSFF	CLEAR EUM SELECTOR FLIP-FLOPS
CLEUUSFF	CLEAR EUU SELECTOR FLIP-FLOPS
CLG1234	CLEAR SDS-ARRAY GATE SIGNALS G1,G2,G3,G4
CLG4	CLEAR SDS4 GATE SIGNAL G4
CLICC	CLEAR COUNTER ICC
CLMSFF	CLEAR M REGISTER SELECTOR FLIP-FLOPS
CLNDCNT	CLEAR COUNTER ND(NUMBER OF DECIMAL DIGITS)
CLNDRR8	CLEAR FLIP-FLOP NDRR8 (NUMBER OF DIVISOR'S 8 BIT RIGHT SHIFTS)
CLNEGO	CLEAR FLIP-FLOP NEGO. (NEGO FLIP-FLOP GENERATES CONTROL NEGO)
CLNEGO03	CLEAR FLIP-FLOP NEGO03
CLNEGO47	CLEAR FLIP-FLOP NEGO47

SIGNAL NAME	SIGNAL FUNCTIONAL DESCRIPTION
CLNEG1	CLEAR FLIP-FLOP NEG1. (NEG1 FLIP-FLOP GENERATES CONTROL SIGNAL NEG1)
CLNEG2	CLEAR FLIP-FLOP NEG2. (NEG2 FLIP-FLOP GENERATES CONTROL SIGNAL NEG2)
CLNEG234	CLEAR FLIP-FLOPS NEG2,NEG3,NEG4
CLNEG3	CLEAR FLIP-FLOP NEG3. (NEG3 FLIP-FLOP GENERATES CONTROL SIGNAL NEG3)
CLNEG4	CLEAR FLIP-FLOP NEG4. (NEG4 FLIP-FLOP GENERATES CONTROL SIGNAL NEG4)
CLSCWORD	CLEAR FLIP-FLOP 'SCWORD' (INDICATES NUMBER OF WORD BEIN TRANSMITTED)
CLUHSFF	CLEAR UH REGISTER SELECTOR FLIP-FLOPS
CLUMSFF	CLEAR UM REGISTER SELECTOR FLIP-FLOPS
CLUQDMR	CLEAR FLIP-FLOP UQDMR (REGISTER UQ LEAST SIGNIFICANT BYTE TO MULTIPLIER RECCRDER)
CLUQSFF	CLEAR REGISTER UQ SELECTOR FLIP-FLOPS
CLUQ65	CLEAR BIT 65 OF REGISTER UQ
CLUREN	CLEAR FLIP-FLOP UREN (UNIT REQUESTS ACCESS TO EXCHANGE NET)
CLUSSFF	CLEAR REGISTER US SELECTOR FLIP-FLOPS
CNTICCDN	COUNT ICC DOWNWARD
CNTICCUP	COUNT ICC UPWARD
CNTND	COUNT COUNTER ND UPWARD
CNTNDDL8UP	COUNT NDDL8 UPWARD
CNTNDRL1DN	COUNT NDRL1 DOWNWARD
CNTNDRL1UP	COUNT NDRL1 UPWARD
CNTNDRL8DN	COUNT NDRL8 DOWNWARD
CNTNDRL8UP	COUNT NDRL8 UPWARD
COMPLWAIT	SUBROUTINE COMPL WAIT.
COMPNEG1	COMPLEMENT CONTROL SIGNAL NEG1
CORREM	CORRECT REMAINDER

SIGNAL NAME	SIGNAL FUNCTIONAL DESCRIPTION
CP	COUNT PULSE
CPRA	ARITHMETIC ORDER COMPARE ALGEBRAIC
CSIGNA	CLEAR FLIP-FLOP SIGNA (SIGN OF OPERAND A)
CSR	CLEAR FLIP-FLOP SR (SIGN OF RESULT)
CUMDX1	CLEAR FLIP-FLOP UMDX1
CUQ133DUS133	CLEAR FLIP-FLOP UQ133DUS133 (REGISTER UQ BITS 1 TO 33 DIRECT OT REGISTER US BITS 1 TO 33)
CUQ912R52UM	CLEAR FLIP-FLOP UQ912R52UM (REGISTER UQ BITS 9 TO 12 SHIFTED RIGHT BY 52 POSITIONS TO UM)
CUSDS1	CLEAR FLIP-FLOP USDS1
CVD	ARITHMETIC ORDER CVD (CONVERT TO DECIMAL)
CVDDIVWAIT	SUBROUTINE CVDDIV WAIT
CVD12AI	CONTROL POINT CVD12 ADVANCE-IN.
CVD13AI	CONTROL POINT CVD13 ADVANCE-IN.
CVF	ARITHMETIC ORDER CVF (CONVERT TO FLOATING POINT)
CVL	ARITHMETIC ORDER CVL (CONVERT TO LONG FIXED POINT)
CY1SFF	CLEAR Y1 SELECTOR FLIP-FLOPS.
CY2SFF	CLEAR Y2 SELECTOR FLIP-FLOPS
CY3SFF	CLEAR Y3 SELECTOR FLIP-FLOPS
CY4SFF	CLEAR Y4 SELECTOR FLIP-FLOPS
DEC	NUMBER TYPE DECIMAL
DEXPGE2	DIFFERENCE OF EXPONENT GREATER THAN OR EQUAL TO 2 IN EXCESS 64 REPRESENTATION
DEXPGTZ	DIFFERENCE OF EXPONENT GREATER THAN ZERO IN EXCESS 64 REPRESENTATION
DEXPGT13	DIFFERENCE OF EXPONENT GREATER THAN 13 IN EXCESS 64 REPRESENTATION
DEXPINR	DIFFERENCE OF EXPONENT IS IN NEGATIVE RANGE
DEXPIPR	DIFFERENCE OF EXPONENT IS IN POSITIVE RANGE

SIGNAL NAME	SIGNAL FUNCTIONAL DESCRIPTION
DEXPLEM2	DIFFERENCE OF EXPONENT IS LESS THAN OR EQUAL TO -2 IN EXCESS 64 REPRESENTATION
DEXPLTM13	DIFFERENCE OF EXPONENT IS LESS THAN -13 IN EXCESS 64 REPRESENTATION
DEXPLTZ	DIFFERENCE OF EXPONENT IS LESS THAN ZERO
DIV	ARITHMETIC ORDER DIV (DIVISION)
DIVIDEWAIT	SUBROUTINE DIVIDE WAIT
DIVID10AO	CONTROL POINT DIVID10 ADVANCE-OUT
DIVID11AI	CONTROL POINT DIVID11 ADVANCE-IN
DIVID7AO	CONTROL POINT DIVID7 ADVANCE OUT
DRTEN	DIVISOR IS TEN
DVBWAIT	SUBROUTINE DVB WAIT
DVB5AI	CONTROL POINT DVB5 ADVANCE-IN
DVB7AI	CONTROL POINT DVB7 ADVANCE-IN
D0DMD	GATE DIVIDEND DIRECT TO MODEL DIVISION
D1DMD	GATE FIRST PARTIAL REMAINDER DIRECT TO MODEL DIVISION
D2DMD	GATE SECOND PARTIAL REMAINDER DIRECT TO MODEL DIVISION
D3DMD	GATE THIRD PARTIAL REMAINDER DIRECT TO MODEL DIVISION
EADEUU	GATE EXPONENT OF OPERAND A DIRECT TO REGISTER EUU
EBDEUM	GATE EXPONENT OF OPERAND B DIRECT TO REGISTER EUM
EQ	EQUAL
EUDIFF	EXPONENT DIFFERENCE
EULDEUU	GATE CONTENTS OF REGISTER EUL DIRECT TO EUU
EULDXI1	GATE CONTENTS OF REGISTER EUL DIRECT TO OUTBUS XT
EULGT21	CONTENTS OF REGISTER EUL IS GREATER THAN 21 IN EXCESS 64 REPRESENTATION
EULGT28	CONTENTS OF REGISTER EUL IS GREATER THAN 28 IN EXCESS 64 REPRESENTATION
EULGT8	CONTENTS OF REGISTER EUL IS GREATER THAN 8 IN EXCESS 64 REPRESENTATION

SIGNAL NAME	SIGNAL FUNCTIONAL DESCRIPTION
EUXDEUM	GATE CONTENTS OF REGISTER EUX DIRECT TO REGISTER EUM
EXDEUX	GATE EXPONENT OF OPERAND X (USED IN POLY) DIRECT TO REGISTER EUX
FA1LFA	GATE FLAGS OF WORD 1 OF OPERAND A TO LEFT HALF OF REGISTER FA
FA2RFA	GATE FLAGS OF WORD 2 OF OPERAND A TO RIGHT HALF OF REGISTER FA
FB1LFB	GATE FLAGS OF WORD 1 OF OPERAND B TO LEFT HALF OF REGISTER FB
FB2RFB	GATE FLAGS OF WORD 2 OF OPERAND B TO RIGHT HALF OF REGISTER FB
FIX	NUMBER TYPE IS FIXED POINT
FIXDIV	CONTROL SEQUENCE DIV WITH-FIXED POINT OPERANDS
FIXDIV14AI	CONTROL POINT DFX14 ADVANCE-IN
FLNOFXWAIT	SUBROUTINE FL-NORM-FX WAIT
FLT	NUMBER TYPE IS FLOATING POINT
FM	FLAG MATCH
GETDEC	GENERATE DECIMAL DIGITES
GETDIG	TRANSFER DECIMAL DIGITS TO REGISTER UQ
GOTOEXIT2	GO TO CONTROL POINT EXIT2
GOTOEXIT3	GO TO CONTROL POINT EXIT3
GT	GREATER THAN
G1	SDS GATE SIGNAL G1
G2	SDS GATE SIGNAL G2
G3	SDS GATE SIGNAL G3
G4	SDS GATE SIGNAL G4
ICCEQZ	CONTENTS OF COUNTER ICC EQUALS ZERO
ICCGTZ	CONTENTS OF COUNTES ICC IS GREATER THAN ZERO
ID	ILLEGAL DATA
ILLDOP	ILLEGAL DECIMAL OPERAND

SIGNAL NAME	SIGNAL FUNCTIONAL DESCRIPTION
ILLOP	ILLEGAL OPERATION CODE
INDFA	GATE INDICATORS DIRECT TO FLAG REGISTER FA
IVDIVR	GATE INSTRUCTION VARIANT BITS DIRECT TO INSTRUCTION VARIANT REGISTER IVR
IV1	INSTRUCTION VARIANT BIT 1
LDDSIGNA	LOAD SIGN OF DECIMAL OPERAND A TO FLIP FLOP SIGNA
LDESDEUL	GATE (LOAD) OUTPUT OF EXPONENT UNIT ADDER TO REGISTER EUL
LDEUM	LOAD REGISTER EUM
LDFAL	LOAD LEFT HALF OF FLAG REGISTER FA
LDFAR	LOAD RIGHT HALF OF FLAG REGISTER FA
LDFSIGNA	LOAD SIGN OF FLOATING OR FIXED POINT OPERAND A TO FLIP-FLOP SIGNA
LDICC	LOAD COUNTER ICC
LDM	LOAD REGISTER M
LDPE	LOAD PARITY ERROR FLIP-FLOP
LDPE1	LOAD PARITY ERROR OF FIRST WORD
LDPE2	LOAD PARITY ERROR OF SECOND WORD
LDPE3	LOAD PARITY ERROR OF THIRD WORD
LDQMS12	LOAD BITS 1 AND 2 OF QUOTIENT BUFFER REGISTERS QM AND QS
LDQMS34	LOAD BITS 3 AND 4 OF QUOTIENT BUFFER REGISTERS QM AND QS
LDQMS56	LOAD BITS 5 AND 6 OF QUOTIENT BUFFER REGISTERS QM AND QS
LDQMS78	LOAD BITS 7 AND 8 OF QUOTIENT BUFFER REGISTERS QM AND QS
LDUH	LOAD REGISTER UH
LDUH47	LOAD BYTES 4 TO 7 OF REGISTER UH
LDUM	LOAD REGISTER UM
LDUQ	LOAD REGISTER UQ
LDUQ03	LOAD BYTES 0 TO 3 OF REGISTER UQ
LDUQ47	LOAD BYTES 4 TO 7 OF REGISTER UQ

SIGNAL NAME	SIGNAL FUNCTIONAL DESCRIPTION
LDUS	LOAD REGISTER US
LFIX	NUMBER TYPE IS LONG FIXED
LHL1UH	SELECT CONTENTS OF REGISTER LH SHIFTED LEFT BY 1 BIT INTO REGISTER UH
LHL4UH	SELECT CONTENTS OF REGISTER LH SHIFTED LEFT BY 4 BITS INTO REGISTER UH
LHL8UH	SELECT CONTENTS OF REGISTER LH SHIFTED LEFT BY 8 BIT INTO REGISTER UH
LHR4UH	SELECT CONTENTS OF REGISTER LH SHIFTED RIGHT BY 4 BITS INTO REGISTER UH
LHR8UH	SELECT CONTENTS OF REGISTER LH SHIFTED RIGHT BY 8 BITS INTO REGISTER UH
LMDM	SELECT CONTENTS OF REGISTER LM DIRECT INTO REGISTER M
LMDUM	SELECT CONTENTS OF REGISTER LM DIRECT INTO REGISTER UM
LMDUQ	SELECT CONTENTS OF REGISTER LM DIRECT INTO REGISTER UQ
LML32UH	SELECT CONTENTS OF REGISTER LM SHIFTED LEFT BY 32 BITS INTO REGISTER UH
LML8UM	SELECT CONTENTS OF REGISTER LM SHIFTED LEFT BY 8 BITS INTO REGISTER UM
LMR4M	SELECT CONTENTS OF REGISTER LM SHIFTED RIGHT BY 4 BITS INTO REGISTER M
LMR8UM	SELECT CONTENTS OF REGISTER LM SHIFTED RIGHT BY 8 BITS INTO REGISTER UM
LM912DM912	SELECT BITS 9 TO 12 OF REGISTER LM DIRECT TO BITS 9 TO 12 OF REGISTER M
LQL1UQ	SELECT CONTENTS OF REGISTER LQ SHIFTED LEFT BY 1 BIT INTO REGISTER UQ
LQL4UQ	SELECT CONTENTS OF REGISTER LQ SHIFTED LEFT BY 4 BITS INTO REGISTER UQ
LQL8UQ	SELECT CONTENTS OF REGISTER LQ SHIFTED LEFT BY 8 BITS INTO REGISTER UQ
LQR1UQ03	SELECT CONTENTS OF BYTES 0 TO 3 OF REGISTER LQ SHIFTED RIGHT BY ONE BIT TO BYTES 0 TO 3 OF REGISTER UQ

LQR4UQ	SELECT CONTENTS OF REGISTER LQ SHIFTED RIGHT BY 4 BITS TO REGISTER UQ
LQR8UQ	SELECT CONTENTS OF REGISTER LQ SHIFTED RIGHT BY 8 BITS TO REGISTER UQ
LS	LOSS OF SIGNIFICANCE
LSDUH	SELECT CONTENTS OF REGISTER LS, DIRECT TO REGISTER UH
LSDUS	SELECT CONTENTS OF REGISTER LS DIRECT TO REGISTER US
LSL8US	SELECT CONTENTS OF REGISTER LS SHIFTED LEFT BY 8 BITS TO REGISTER US
LSR8US	SELECT CONTENTS OF REGISTER LS SHIFTED RIGHT BY 8 BITS TO REGESTER US
LT	LESS THAN
MDY1	SELECT CONTENTS OF M DIRECT TO Y1
MDY4	SELECT CONTENTS OF M DIRECT TO Y4
MDDINT	GATE CONTENTS OF BITS 10 TO 12 OF REGISTER M TO DIVISOR INTERVAL SESECT
MEPGATE	GATE MULTIPLY-EXTENDED-PRECISION BITS
MEPLATCH	LATCH MULTIPLY-EXTENDED-PRECISON BITS
ML1Y4	SELECT CONTENTS OF REGISTER M SHIFTED LEFT BY 1 BIT TO Y4
ML2Y3	SELECT CONTENTS OF REGISTER M SHIFTED LEFT BY 1 BIT TO Y3
ML3Y3	SELECT CONTENTS OF REGISTER M SHIFTED LEFT BY 3 BITS TO Y3
ML4Y2	SELECT CONTENTS OF REGISTER M SHIFTED LEFT BY 4 BITS TO Y2
ML5Y2	SELECT CONTENTS OF REGISTER M SHIFTED LEFT BY 5 BITS TO Y2
ML6Y1	SELECT CONTENTS OF REGISTER M SHIFTED LEFT BY 6 BITS TO Y1
ML7Y1	SELECT CONTENTS OF REGISTER M SHIFTED LEFT BY 7 BITS TO Y1
MPY	ARITHMETIC ORDER MULTIPLY

SIGNAL NAME	SIGNAL FUNCTIONAL DESCRIPTION
MPYSTOP	MULTIPLICATION LOOP STOP
MPY7AI	CONTROL POINT MPY7 ADVANCE-IN
MPY5AI2	CONTROL POINT MPY5 ADVANCE-IN 2.
NDDL8DNDRL8	GATE CONTENTS OF COUNTER NDDL8 DIRECT TO COUNTER NDRL8
NDRL1EZ	CONTENTS OF COUNTER NDRL1 EQUALS ZERO
NDRL4	DIVISOR HAS BEEN SHIFTED LEFT ONCE BY FOUR BITS
NDRL8EZ	CONTENTS OF COUNTER NDRL8 EQUALS ZERO
NDRR8	DIVISOR HAS BEEN SHIFTED RIGHT BY 8 BITS ONCE
NEGATE	INPUT TO PROCEDURE NEGATE
NEGR	NEGATIVE REMAINDER
NEXTDIV	ENTER DIVISICN LOCP AT ENTRY NEXTDIV FOR ANOTHER PASS
NORMUQ	NORMALIZE UQ
NTDNTR	GATE NUMBER TYPE BITS DIRECT TO REGISTER NTR
OV	OVERFLOW
PDY4	SELECT OUTPUT OF PROPAGATION LOGIC DIRECT TO Y4
PERR	PARITY ERROR
POLY	ARITHMETIC ORDER POLY
POLYCO	POLY OPERATION CONTINUED
QASS	QUOTIENT ASSIMILATION
QBDUQH	GATE QUOTIENT BUFFER REGISTERS QM AND QS TO REGISTER UQ AND UH
RESCALEREM	RESCALE REMAINDER
RESTODIV	RESTORING DIVISION
RETURNCVD	RETURN TO CONTROL SEQUENCE CVD
SBYONE	SET FLIP-FLOP BYONE
SCALFIXDDR	SCALE FIXED-POINT DIVISOR
SDB	SIGN DETERMINATION BYPASS
SDODMD	SET SELECTOR FLIP-FLOP DODMD

SIGNAL NAME	SIGNAL FUNCTIONAL DESCRIPTION
SD1DMD	SET SELECTOR FLIP-FLOP D1DMD
SD2DMD	SET SELECTOR FLIP-FLOP D2DMD
SD3DMD	SET SELECTOR FLIP-FLOP D3DMD
SEADEUU	SET SELECTOR FLIP-FLOP EADEUU
SEBDEUM	SET SELECTOR FLIP-FLOP EBDEUM
SETDECSIGN	SET SELECTOR FLIP-FLOP DECSIGN
SETFLAGAI	CONTROL SEQUENCE SFBIN ADVANCE-IN (SET FLAGS CONTROL SUBSEQUENCE ADVANCE-IN)
SETM64ONE	SET BIT 64 OF REGISTER M TO ONE STATE.
SETSIGN	SET SIGN (ADVANCE-IN SIGNAL TO PROCEDURE SUBSEQUENCE SETSIGN)
SETUQ03ONE	SET BYTES 0 TO 3 OF REGISTER UQ TO ALL '1' STATE
SETUQ33ONE	SET BIT 33 OF REGISTER UQ TO '1' STATE
SETUQ33ZERO	SET BIT 33 OF REGISTER UQ TO '0' STATE
SETUQ47ONE	SET BYTES 4 TO 7 OF REGISTER UQ TO ALL '1' STATE
SEUUEZ	SET CONTENTS OF REGISTER EUU EQUAL TO ZERO IN EXCESS 64 REPRESENTATION
SEUCTRDIR	SET FLIP-FLOP EUCTRDIR (EUL COUNTER DIRECTION)
SEUDIFF	SET FLIP-FLOP EUDIFF TO '1' STATE
SEULALLONE	SET REGISTER EUL TO ALL '1' STATE
SEULALLZERO	SET REGISTER EUL TO ALL '0' STATE
SEULEQ14	SET CONTENTS OF REGISTER EUL EQUAL TO 14
SEULEQ8	SET CONTENTS OF REGISTER EUL EQUAL TO 8
SEUMEQ14	SET CONTENTS OF REGISTER EUM EQUAL TO 14
SEUMEZ	SET CONTENTS OF REGISTER EUM EQUAL TO ZERO IN EXCESS 64 REPRESENTATION
SEUMZERO	SET BIT 1 OF REGISTER EUM TO '1' STATE
SEUUEZ	SET CONTENTS OF REGISTER EUU EQUAL TO ZERO IN EXCESS 64 REPRESENTATION
SEUUZERO	SET BIT 1 OF REGISTER EUU TO '1' STATE

SIGNAL NAME	SIGNAL FUNCTIONAL DESCRIPTION
SFIX	SHORT FIXED POINT
SG1CLG2G3G4	SET SDS GATE SIGNALS G1='1', G2=G3=G4='0'
SG1G2G3G4	SET SDS GATE SIGNELS G1=G2=G3=G4='1'
SICCCNTDN	SET ICC COUNTER DIRECTION FLIP-FLOP TO, DOWNWARD COUNTING STATE
SICCCNTUP	SET ICC COUNTER DIRECTION FLIP-FLOP TO UPWARD COUNTING STATE
SICCEQ7	SET CONTENTS OF COUNTER ICC EQUAL TO 7
SICCEQ8	SET CONTENTS OF COUNTER ICC EQUAL TO 8
SID	SET ILLEGAL DATA FLIP-FLOP ID
SIGNACDNEG1	GATE COMPLEMENT OF FLIP-FLOP SIGNA TO FLIP-FLOP NEG1
SIGNADSR	GATE CONTENTS OF FLIP-FLOP SIGNA TO FLIP-FLOP SR
SIGNB	SIGN OF OPERAND B
SIGNBMPY	SIGN OF PERAND B DURING CONTROL SEQUENCE MPY (TEMPORARY STORAGE)
SINJECTSIGN	SET FLIP-FLOP INJECTSIGN
SLHL1UH	SET SELECTOR FLIP-FLOP LHL1UH TO '1' STATE
SLHL4UH	SET SELECTOR FLIP-FLOP LHL4UH TO '1' STATE
SLHL8UH	SET SELECTOR FLIP-FLOP LHL8UH TO '1' STATE
SLHL4UH	SET SELECTOR FLIP-FLOP LHR4UH TO '1' STATE
SLHR8UH	SET SELECTOR FLIP-FLOP LHR8UH TO '1' STATE
SLMDM	SET SELECTOR FLIP-FLOP LMDM TO '1' STATE
SLMDUM	SET SELECTOR FLIP-FLOP LMDUM TO '1' STATE
SLMDUQ	SET SELECTOR FLIP-FLOP LMDUQ TO '1' STATE
SLMDUQ03	SET SELECTOR FLIP-FLOP LMDUQ03 TO '1' STATE
SLMDUQ47	SET SELECTOR FLIP-FLOP LMDUQ47 TO '1' STATE
SLML32UH	SET SELECTOR FLIP-FLOP LML32UH TO '1' STATE
SLML8UM	SET SELECTOR FLIP-FLOP LML8UM TO '1' STATE
SLMR4M	SET SELECTOR FLIP-FLOP LMR4M TO '1' STATE

SIGNAL NAME	SIGNAL FUNCTIONAL DESCRIPTION
SLMR8UM	SET SELECTOR FLIP-FLOP LMR8UM TO '1' STATE
SLM912DM912	SET SELECTOR FLIP-FLOP LM912DM912 TO '1' STATE
SLQL1UQ	SET SELECTOR FLIP-FLOP LQL1UQ TO '1' STATE
SLQL4UQ	SET SELECTOR FLIP-FLOP LQL4UQ TO '1' STATE
SLQL8UQ	SET SELECTOR FLIP-FLOP LQL8UQ TO '1' STATE
SLQL8UQ	SET SELECTOR FLIP-FLOP LQL8UQ TO '1' STATE
SLQR1UQ03	SET SELECTOR FLIP-FLOP LQR1UQ03 TO '1' STATE
SLQR4UQ	SET SELECTOR FLIP-FLOP LQR4UQ TO '1' STATE
SLQR8UQ	SET SELECTOR FLIP-FLOP LQR8UQ TO '1' STATE
SLSDUH	SET SELECTOR FLIP-FLOP LSDUH TO '1' STATE
SLSDUS	SET SELECTOR FLIP-FLOP LSDUS TO '1' SATE
SLSL8US	SET SELECTOR FLIP FLOP LSL8US TO '1' STATE
SLSR8US	SET SELECTOR FLIP-FLOP LSR8US TO '1' STATE
SMDY1	SET SELECTOR FLIP-FLOP MDY1 TO '1' STATE
SMDY4	SET SELECTOR FLIP-FLOP MDY4 TO '1' STATE
SML1Y4	SET SELECTOR FLIP-FLOP ML1Y4 TO '1' STATE
SML3Y3	SET SELECTOR FLIP-FLOP ML3Y3 TO '1' STATE
SMYRECODON	SET CONTROL FLIP-FLOP MYRECOD(MULTIPLY RECODER)
SNDDL8UP	SET NDDL8 COUNTER DIRECTION FLIP-FLOP TO UPWARD COUNTING STATE
SNDRL1DN	SET NDRL1 COUNTER DIRECTION FLIP-FLOP TO DOWNWARD COUNTING STATE
SNDRL1UP	SET NDRL1 COUNTER DIRECTION FLIP-FLOP TO UPWARD COUNTING STATE
SNDRL8DN	SET NDRL8 COUNTER DIRECTION FLIP-FLOP TO DOWNWARD COUNTING STATE
SNDRL8UP	SET NDRL8 COUNTER DIRECTION FLIP-FLOP TO UPWARD COUNTING STATE
SNEGO	SET SELECTOR FLIP-FLOP NEGO
SNEGO03	SET SELECTOR FLIP-FLOP NEGO03

SIGNAL NAME	SIGNAL FUNCTIONAL DESCRIPTION
SNEG047	SET SELECTOR FLIP-FLOP NEG047
SNEG1	SET SELECTOR FLIP-FLOP NEG1
SNEG2	SET SELECTOR FLIP-FLOP NEG2
SNEG4	SET SELECTOR FLIP-FLOP NEG4
SOV	SET OVERFLOW FLIP-FLOP OV
SPDY4	SET SELECTOR FLIP-FLOP PDY4
SQBDUQH	SET SELECTOR FLIP-FLOP QBDUQH
SRCDNEG1	GATE COMPLEMENT OF FLIP-FLOP SR TO FLIP-FLOP NEG1
SSIGNA	SET FLIP-FLOP SIGNA (SIGN OF OPERAND A)
SSCWORD	SET FLIP-FLOP SCWORD
SSR	SET FLIP-FLOP SR (SIGN OF RESULT)
STENDY1	SET SELECTOR FLIP-FLOP TENDY1
SUB	ARITHMETIC ORDER SUBTRACT
SEULALLZERO	SET ALL BITS OF EUL REGISTER TO '0' STATE
SUHDM	SET SELECTOR FLIP-FLOP UHDM
SUHDUS	SET SELECTOR FLIP-FLOP UHDUS
SUMDX1	SET GATING SIGNAL FLIP-FLOP UMDX1
SUQDMR	SET GATING SIGNAL FLIP-FLOP UQDMR
SUQDUM	SET SELECTOR FLIP-FLOP UQDUM
SUQ33DUS33	SET SELECTOR FLIP-FLOP UQ33DUS33
SUQ33ONE	SET BIT 33 OF REGISTER UQ TO '1' STATE
SUQ33ZERO	SET BIT 33 OF REGISTER UQ TO '0' STATE
SUQ47ONE	SET BIT 47 OF REGISTER UQ TO '1' STATE
SUQ58DM6164	SET SELECTOR FLIP-FLOP UQ58DM6164
SUQ912R52UM	SET SELECTOR FLIP-FLOP UQ912R52UM
SUSDS1	SET GATING SIGNAL FLIP-FLOP USDS1
SUS9	SET BIT 9 OF REGISTER US TO '1' STATE

SIGNAL NAME	SIGNAL FUNCTIONAL DESCRIPTION
TIO	'TRANSFER INFORMATION OUT' TO EXCHANGE NET, PULSE
T4DLS	GATE SDS4 OUTPUT T4 DIRECT INTO REGISTER LS
UHDLH	GATE CONTENTS OF REGISTER UH DIRECT INTO REGISTER LH
UHDM	SELECT CONTENTS OF REGISTER UH DIRECT TO REGISTER M
UHDUS	SELECT CONTENTS OF REGISTER UH DIRECT TO REGISTER US
UHDY1	SELECT CONTENTS OF REGISTER UH DIRECT TO Y1
UQ1DUS1	SELECT BIT 1 OF UQ DIRECT TO BIT 1 OF US
UQ13DXT	SELECT BYTES 1 TO 3 OF REGISTER UQ DIRECT TO OUTBUS XT
UQ33DUS33	SELECT BIT 33 OF REGISTER UQ DIRECT TO BIT 33 OF US
UQ47DXT	SELECT BYTES 4 TO 7 OF REGISTER UQ DIRECT TO OUTBUS
UQ58DM6164	SELECT BITS 5 TO 8 OF REGISTER UQ DIRECT TO BITS 61 TO 64 OF REGISTER M
UQ912R52UM	SELECT BITS 9 TO 12 OF REGISTER UQ SHIFTED RIGHT BY 52 BITS TO REGISTER UM
UQ916EZ	CONTENTS OF BITS 9 TO 16 OF REGISTER UQ EQUALS ZERO
UREN	UNIT REGUESTS THE EXCHANGE NET
VDUH3	SELECT BITS 1U TO 18 OF V-BUS INTO BYTE 0 OF REGISTER UH
VDUH13	SELECT IBTS 21-28,31-38 AND 41-48 OF V-BUS INTO BYTES 1 TO 3 OF REGISTER UH
VDUH47	SELECT BITS 11-18,21-28,31-38,41-48 OF V-BUS TO BYTES 4 TO 7 OF REGISTER UH
VDUQ0	SELECT BITS 11-18 OF V-BUS TO BYTES 0 OF REGISTER UQ
VDUQ13	SELECT BITS 21-28,31-38,41-48 TO BYTES 1 TO 3 OF REGISTER UQ
VUDQ47	SELECT BITS 11-18,21-28,31-38,41-48 OF V-BUS TO BYTES 4 TO 7 OF REGISTER UQ
VIN3AI	CONTROL POINT VIN3 ADVANCE-IN
VIN3AO	CONTROL POINT VIN3 ADVANCE-OUT
VIN4AI	CONTROL POINT VIN4 ADVANCE-IN
VIN5AI	CONTROL POINT VIN5 ADVANCE-IN

SIGNAL NAME	SIGNAL FUNCTIONAL DESCRIPTION
WAITNORMUQ	SUBROUTINE NORMUQ WAIT
XOUTAI	CONTROL SEQUENCE X-OUT ADVANCE-IN
XTDES	GATE EXITBUS XT CONTENTS DIRECT TO EXCHANGE NET
ZERODIVND	ZERO DIVIDEND
Z4DLM	GATE SDS4 OUTPUT Z4 DIRECT INTO REGESTER LM
UHTUQWAIT	SUBROUTINE UHTUQ WAIT
UH916EZ	CONTENTS OF BITS 9 TO 12 OF REGISTER UH EQUALS ZERO
UN	UNDERFLOW
UQDLQ	GATE CONTENTS OF UQ DIRECT TO LQ
UQDMR	GATE UQ DIRECT TO MULTIPLIER RECODER
UQDUM	SELECT CONTENTS OF UQ DIRECT TO REGISTER UM
UQODXT	SELECT CONTENTS OF BYTE 0 OF REGISTER UQ TO OUTBUS XT

BIBLIOGRAPHY

Atkins, D. E. "Illiac III Computer System Manual: Arithmetic Units," Volume 1, Department of Computer Science Report No. 366, University of Illinois at Urbana-Champaign, December 1969.

Atkins, D. E. "Design of the Arithmetic Units of Illiac III: Use of Redundancy and Higher-Radix Methods," Department of Computer Science Report No. 333, University of Illinois at Urbana-Champaign, May 1969.

Goyal, L. N., Koo, P. L. and Atkins, D. E. "Arithmetic Unit of Illiac III: Simulation and Logical Design--Part II," Department of Computer Science Report No. 418, University of Illinois at Urbana-Champaign, November 1970.

Krabbe, S. P. "A Discussion of Illiac III Processor-Unit Communication via the Exchange Net," Department of Computer Science File No. 790, University of Illinois at Urbana-Champaign, March 1969.

Krabbe, S. P. "Illiac III Check-Out and Maintenance Manual: Arithmetic Unit 0," Department of Computer Science File No. 872, University of Illinois at Urbana-Champaign (in preparation).

Martin, R. G. "Standardization of Control Point Realization," Department of Computer Science Report No. 400 (M.S. Thesis), University of Illinois at Urbana-Champaign, May 1970.

Nordmann, B. J. and Atkins, D. E. "Supplementary Material on Control Point Logic Design," classnotes for CS 294, Fall 1969-70, University of Illinois at Urbana-Champaign.

APPENDIX

A.0 Introduction

This section is a reproduction* of Section 3 entitled 'CONTROL POINT - A Building Block Approach' of DCS Report No. 400 by Mr. R. G. Martin. This report is a reproduction of his M. S. thesis entitled 'Standardization of Control Point Realization'.

This section describes, in detail, the task stage and sequence stage and the timing stage of a basic Control Point and their corresponding logic design. Variants of the basic control point are also discussed.

---

* Reproduced with the author's permission.

A. CONTROL POINT - A BUILDING BLOCK APPROACH

A.1 Description

Pseudo-asynchronous control is based on the concept that tasks be performed in controlled steps or in irregular time intervals dependent on the execution time of these tasks. This type of control has been implemented at Illinois by the use of logical circuits called control point. The control point originally developed as a modification of the Illiac II "speed independent" control circuits by Gilles, Robertson, and Swartwout (1961). Generally speaking, each control step is performed by one control point. In many cases, the control point may be used to implement several similar control steps from different parts of a sequence and will return to that part of the sequence from which it was called.

The basic idea behind a control point is that it initiates some operation by turning on a given set of control lines. These control lines will remain on until either a certain length of time has passed, or a reply has been received indicating that the operation has been completed. As these control lines are turned off, an advance signal is initiated which will activate the next control point.

The use of control point, therefore, provides an ordered scheme for the assignment of specific tasks being performed in controlled steps. In other words, the control point renders to the logical designer a building block approach or technique in the design of pseudo-asynchronous control.

The evolution of the control point design for Illiac III has been long and tortuous. The investigation and applications of several control

point configurations have been made by Atkins and Nordmann (1969). The control point configuration presented in this paper consists of two stages: a task stage and a timing stage.

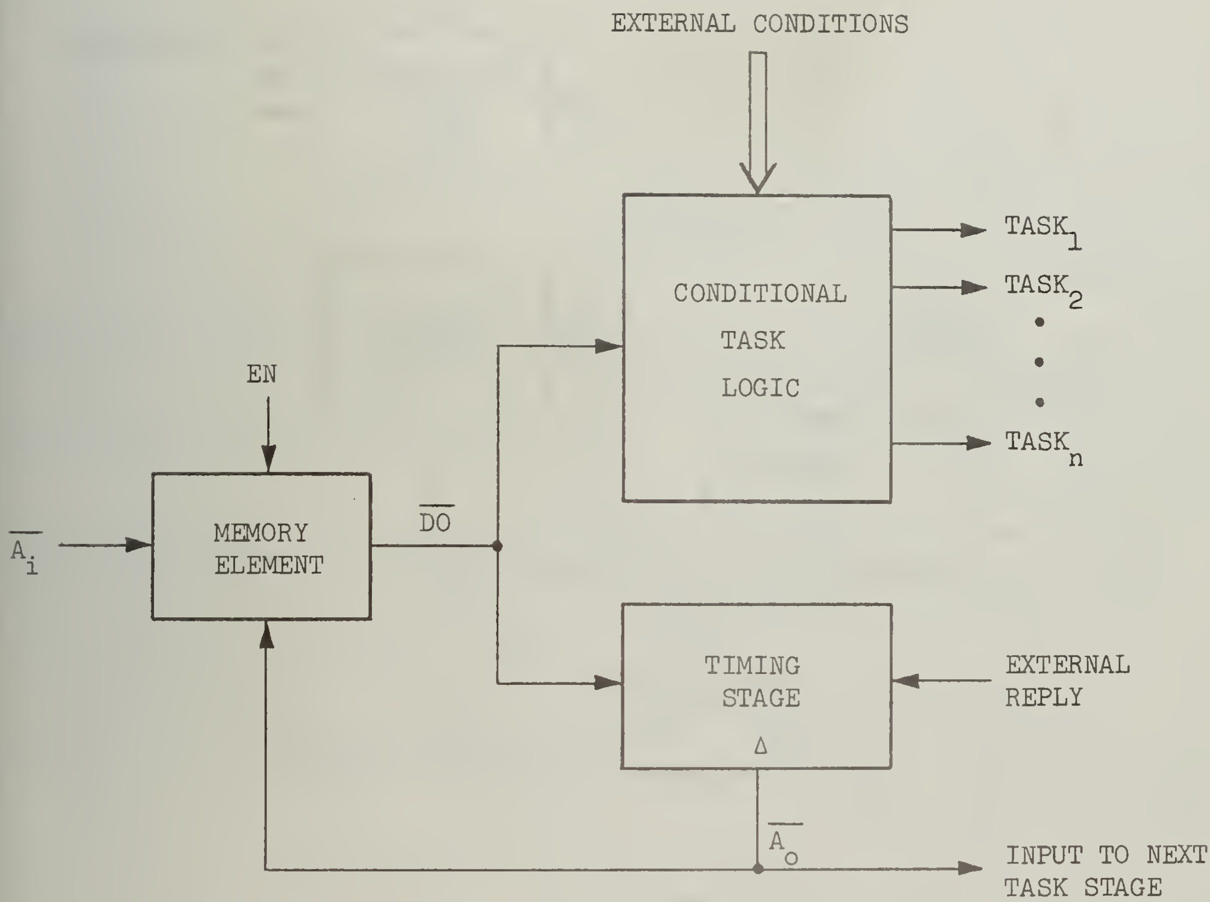
A.1.1 Task Stage

The function of the task stage is to perform certain operations on various hardware: i.e., gates are operated, flip-flops are set, counters incremented or other control points called. These operations may be conditional upon status conditions. Consider the block diagram of typical task stage logic as shown in Figure A.1.1.1. The advance in the $\overline{A}_i$ is connected to an adjacent stage logic. When this line drops to "0", the memory element is set and the task logic is said to be primed. When $\overline{A}_i$ returns to "1", the task activation signal, $\overline{DO}$, becomes "0" provided that the enable line, EN, is at "1". The task stage logic is now said to have been initiated.

The $\overline{DO}$ signal from the memory box is one input to the conditional task logic, while the other inputs to this logic are external conditions appropriate to that task stage. Typical examples of these conditions are the outputs of special condition detect logic at the output of certain registers, or the outputs of status flip-flops, etc.

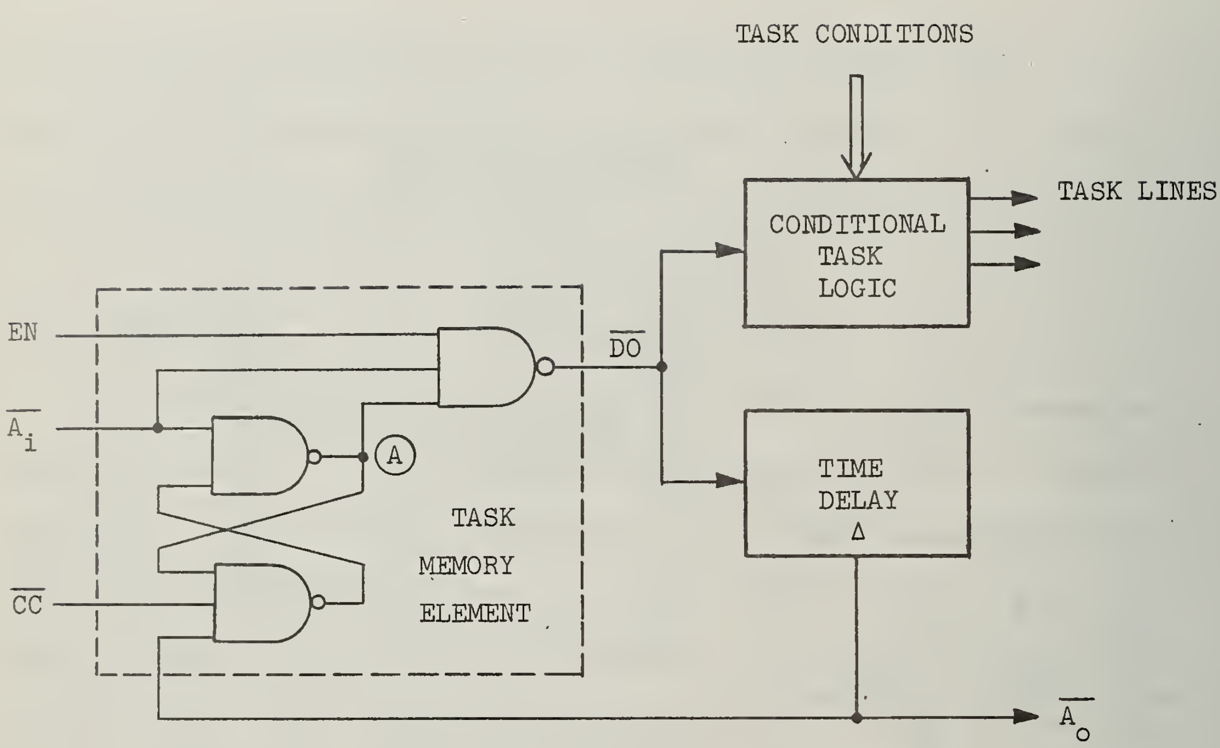
The $\overline{DO}$ signal also activates the timing stage which, after a selectable duration, causes the advance out line, $\overline{A_O}$, to go to "0". This action will reset the memory element, thus turning off the task element, and can be used to also prime and initiate the next succeeding task stage.

Figure A.1.1.2 illustrated the most elementary task stage configuration and an explanatory timing diagram. In this case the timing stage will consist of an internal timing model which will be explained later in this paper.

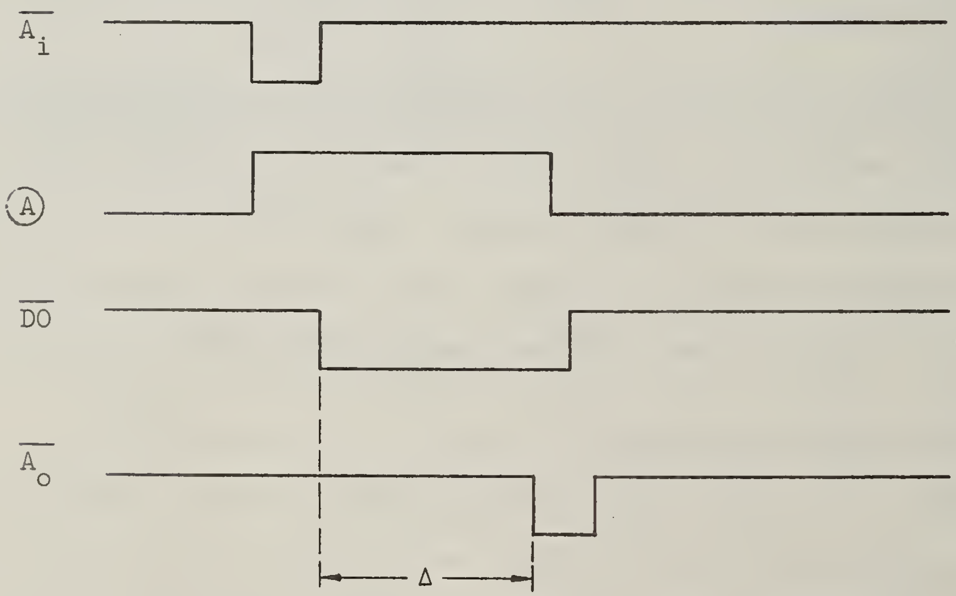


Δ = f (internal time delay or external reply signal)

Figure A.1.1.1. Block Diagram of Task Stage Logic



TIMING DIAGRAM:



Assume that EN = 1 and $\overline{CC}$ = 1

Figure A.1.1.2. Most Elementary Task Stage Configuration

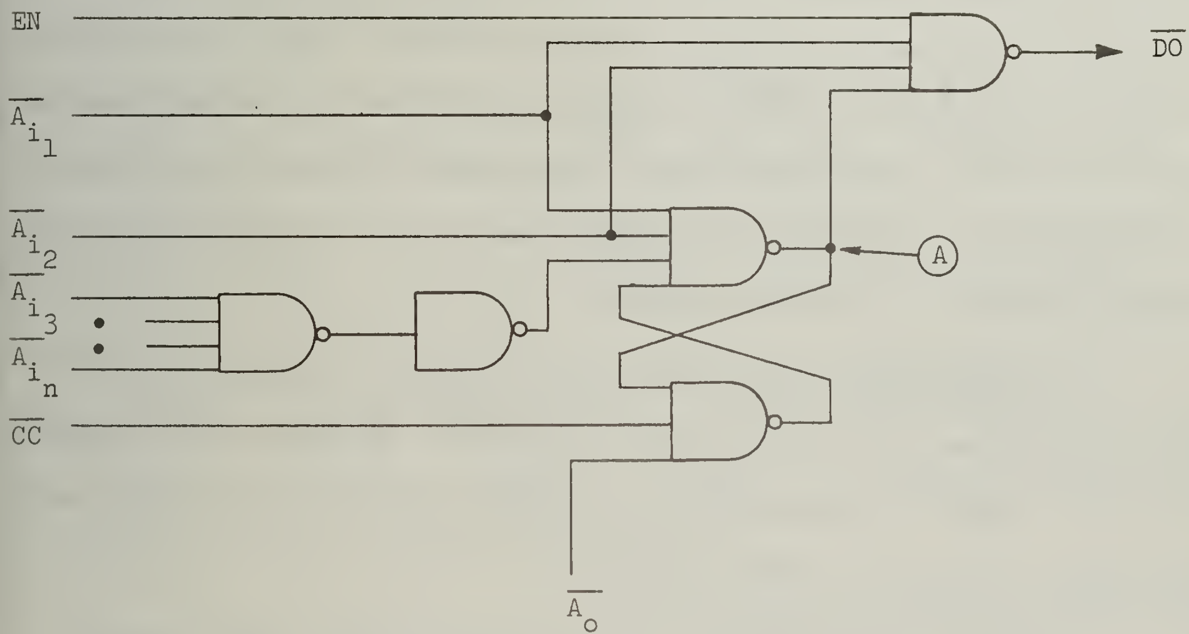


Figure A.1.1.3. Multiple "Advance In" Input to Memory Element

No detailed logic is shown in the conditional task logic box since the configuration is highly variable from task stage to task stage. The most elementary configuration would be a direct connection of the $\overline{DO}$ (perhaps through an invertor or line driver) to the task lines.

The enable input,EN,offers a facility for inhibiting the operation of the task stage. If this input is held at "0", the control sequence will stop when the memory element of the inhibited stage is initiated. As EN returns to "1", the normal operation of $\overline{DO}$ will resume. This input may then be used to either inhibit a control sequence conditional upon an asynchronous signal or serve as a maintenance stop. A maintenance stop may be performed either manually as a function of a control panel switch, or automatically by the use of a diagnostic testing routine.

The <u>common-clear</u> input, $\overline{CC}$, provides a means by which the task stage memory element can be set to the "off" state. Typically the $\overline{CC}$ input of all the task stages of a routine(s) are connected together and "initialized" at the start of a specific control sequence.

It may be found desirable in some instances that a task stage be initiated from more than one advance in line. Figure A.1.1.3 illustrates a method for implementing multiple advance in signals to a memory element. The $\overline{A}_i$ inputs are quiescently "1". When any one drops to "0", the memory element flip-flop is set such that A = "1". As long as any $\overline{A}_i$ or the EN input is "0" the $\overline{DO}$ signal remains at "1", but when they return to "1" $\overline{DO}$ drops to activate the task logic. The duration of the $\overline{A}_{i_3} \cdot - \cdot \overline{A}_{i_n}$ signals must be long enough to allow the memory flip-flops to set (i.e., compensate for the added propagation delay caused by the two additional NANDs).

A.1.2 Timing Stage

It is the function of this stage to provide a delay of the $\overline{DO}$ signal for a time period necessary to complete all the tasks of a given task stage. Associated with this time delay is the logic required for the generation of a reply signal $\overline{A_O}$ used to reset the task memory flip-flop, which turns off the task lines, and drives the next sequence stage logic which primes and initiates the next task stage.

In many cases this time delay is generated by an internal delay element which provides a timing model of the actual task. Figure A.1.2.1 illustrates a timing stage configuration and an explanatory timing diagram. As $\overline{DO}$ goes to "0" (A) , the output of NAND I, goes to "1". The <u>advance out</u> line, $\overline{A_O}$, will be delayed from going to "0" by the amount of time required for the RC-network at the (B) input of NAND II to charge to the logical "1" threshold. When this threshold is reached, $\overline{A_O}$ drops to "0" causing the task memory to be reset and therefore $\overline{DO}$ will return to "1".

The following design information on this circuit configuration was obtained by the use of basic circuit transformation techniques. A simplified model of NAND II and the equivalent circuits used in this analysis are shown in Figure A.1.2.2. The logic elements used in this design were 54/74 series TTL and the following assumptions have been made.

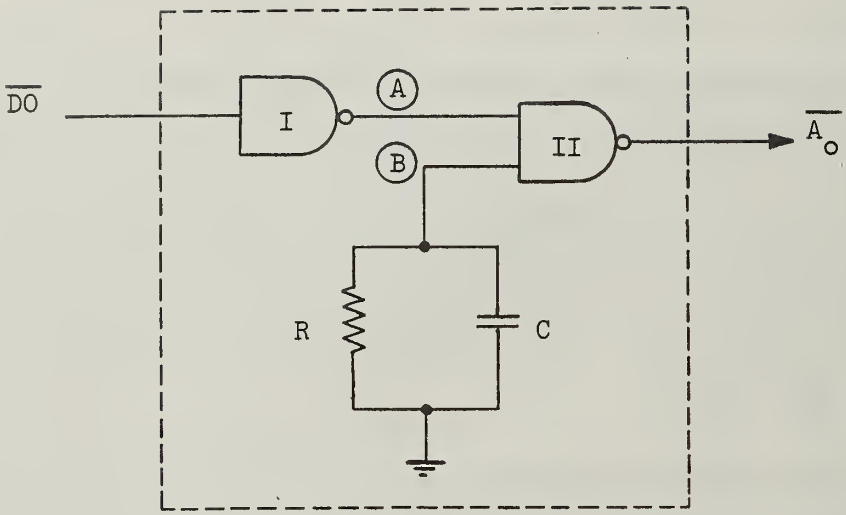
V_{cc} = 5.0 volts

$V_{THRESHOLD} = V_{(B)\,"1"}$ = 1.4 volts (midpoint of uncertainty range)

R_1 = 4 K Ω (value given in T.I.-TTL handbook)

R_2 = 8.2 K Ω (determined empirically)

C = 100 pfd.



TIMING DIAGRAM:

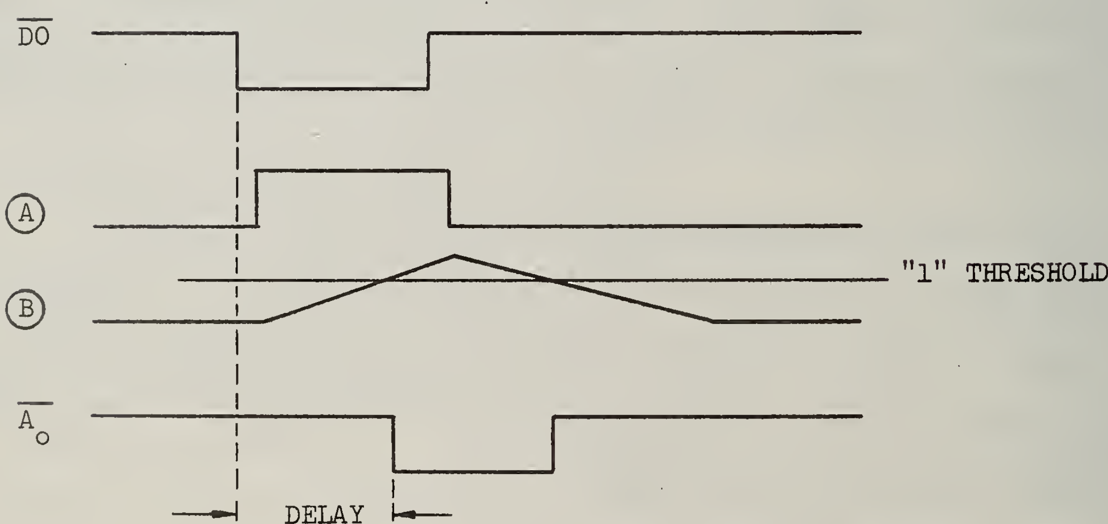


Figure A.1.2.1. Timing Stage Using Internal Delay Element

Using the Thévenin equivalent circuit, the timing model has been reduced to a low-pass RC-network where

$$V_{TH} = V_{CC}\left(\frac{R_2}{R_1 + R_2}\right) = 5\left(\frac{8.2}{12.2}\right) = 3.36 \text{ volts}$$

$$R_{TH} = \frac{R_1 \quad R_2}{R_1 + R_2} = 2.69 \text{ K } \Omega$$

The delay time is related then to the equation for the charging of capacitor C

$$V_C = V_{TH} \left(1 - e^{-t/(R_{th}C)} \right)$$

If we assume that V_C is initially zero and that the delay ends when V_C = threshold = 1.4 volts, then

$$1.4 = 3.36 \left(1 - e^{-t/2.69 \times 10^{-7}}\right)$$

$$t = (.538 \times 2.69 \times 10^{-7}) = 145 \text{ nS}$$

The recovery time or the time required to discharge C is determined solely by R_2 and C. The discharge time can then be expressed by the equation:

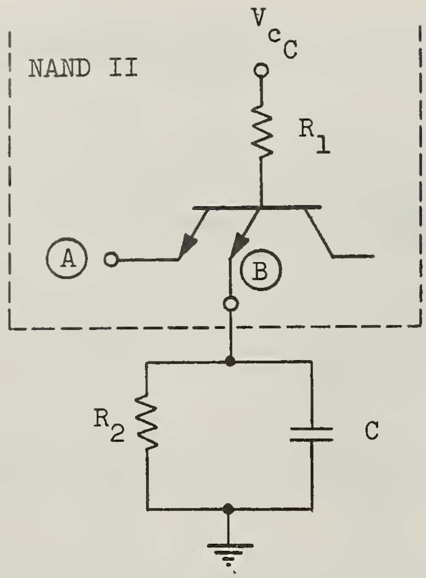
$$V_C = V_{TH} \left(e^{-t/R_2C}\right)$$

If we assume that V_C is initially equal to V_{TH} and that C is considered to be discharged when $V_C = .1V_{TH}$, then

$$.336 = 3.36 \left(e^{-t/8.2 \times 10^{-7}}\right)$$

$$t = (2.3 \times 8.2 \times 10^{-7}) = 1885\text{nS}$$

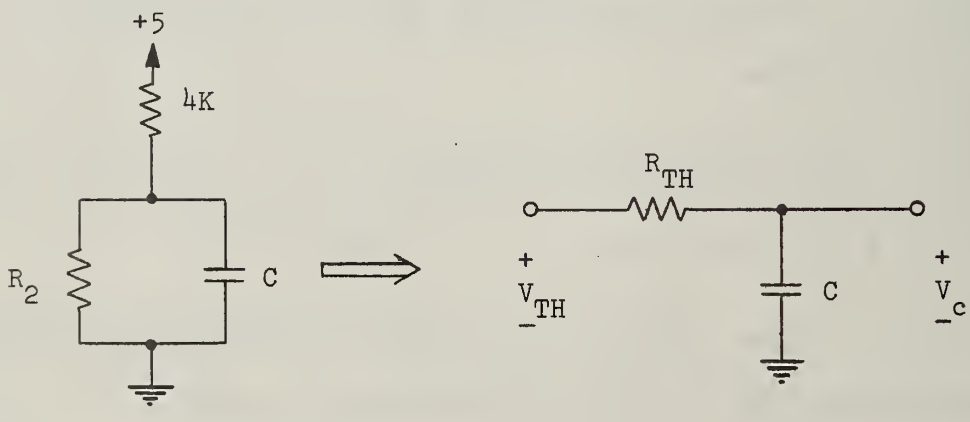
It can then be seen from the above calculation that approximately 13 delay times must elapse between the application of input pulses at A to



NAND II is typically SN7400N

EQUIVALENT CIRCUIT

THEVENIN'S EQUIVALENT



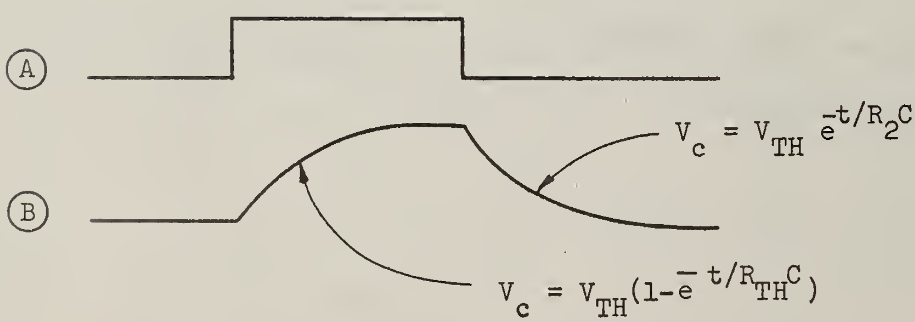


Figure A.1.2.2 Timing Stage Equivalent Circuits

avoid interaction. If this constraint hinders circuit performance it may be greatly reduced by the addition of a diode as shown in Figure A.1.2.3. This diode, a USD25 is a HP2800 hot carrier device with a low forward drop and junction capacitance, is used to discharge C when A goes to "0". With the addition of this diode, only 1.5 delay times are required between input pulses or the timing stage can be pulsed as soon as it has completed its delay function.

Empirical tests have indicated a design center value for R_2 of 8.2KΩ with upper and lower limits of 12K and 6.2K respectively. Using $R_2 = 8.2K\Omega$, the following was found to hold:

Delay - 1.5nS/pFd.

Variation of $\pm$ .5v in V_{CC} - $\pm$ 15%

Variation between IC packages - $\pm$ 15%

Variation due to $R_2 = 6.2K \rightarrow 12K$ - 8%

The delay time of the timing stage could also be a direct function of an external replay signal as has been shown in Figure A.1.2.4. In this case the RC-network has been replaced by the reply line, GO.

Once again as in the previous example, as $\overline{DO}$ goes to "0", the output of NAND I goes to "1". The advance out line $\overline{A}_o$ will not go to "0" until GO has been set to "1". If GO were "1" as $\overline{DO}$ goes to "0", the delay time would then depend on the propagation time of NAND I and NAND II and the amount of time required for the task flip-flop to be reset by $\overline{A}_o$.

The control point is then the combination of a task stage and a timing stage with the block diagram and circuit configuration being represented in Figure A.1.2.5.

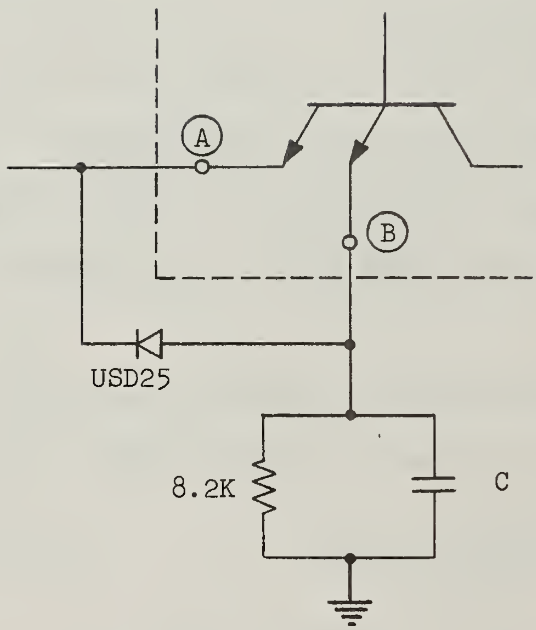


Figure A.1.2.3 Delay Circuit with Diode to Enhance Recovery

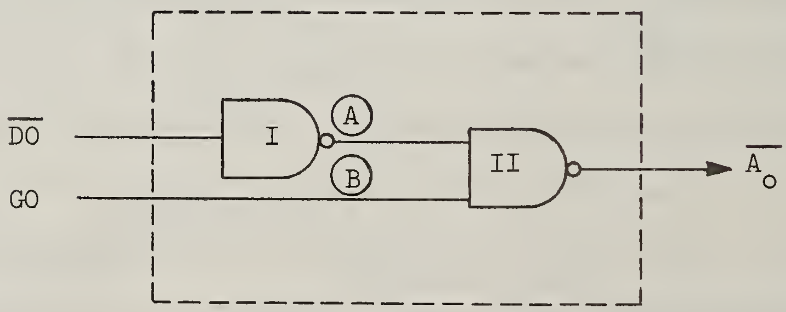
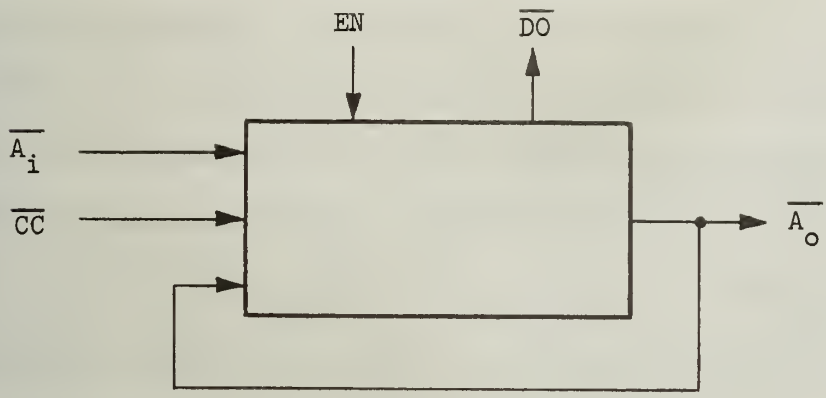
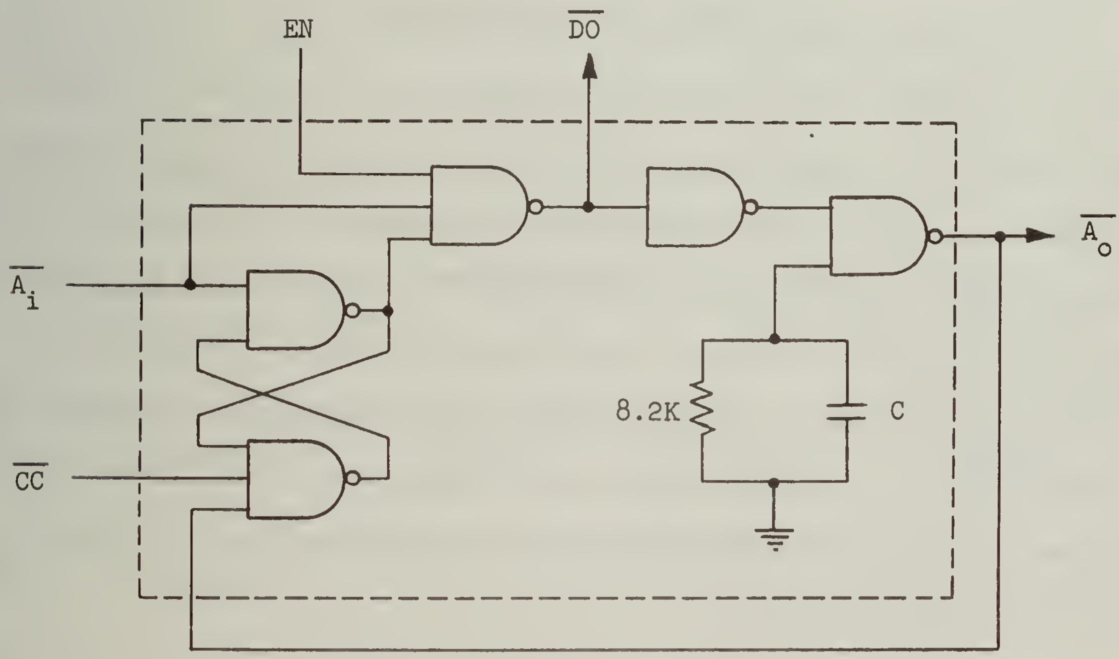


Figure A.1.2.4 Timing Stage Using External Reply





where C = f (time delay required)

Figure A.1.2.5 Control Point Block Diagram and Circuit Configuration

A.2 Sequence Stage

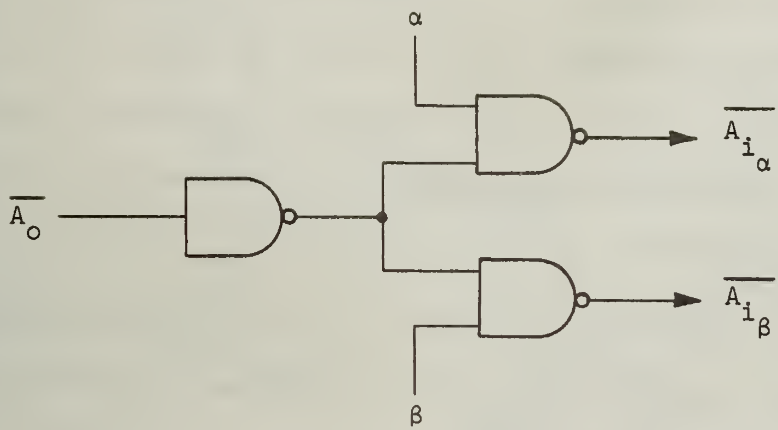
Interspersed with the control points used to implement the asynchronous controlled steps is the sequence stage. The function of the stage is basically one of selection or steering. That is, as a control point completes its specific task the sequence stage makes the decision as to which control point(s) to initiate next.

The most elementary example of sequence stage logic is merely a wire connecting the $\overline{A}_o$ output of one control point to the $\overline{A}_i$ input of the next control point.

In the case where two or more control points are to be called concurrently, the sequence stage logic will be AND'ed as a function of $\overline{A}_o$ and external conditions. Figure A.2.1 illustrates an example of a two-way branch. The conditions, α and β, determine where the $\overline{A}_o$ pulse from the previous control point is directed. It is important that these conditions are set prior to the arrival of the $\overline{A}_o$ pulse, therefore, it is advisable that α or β not be determined by the action of the control point which generates the $\overline{A}_o$ pulse which they steer. It should be noted that at least one condition must be true or else the $\overline{A}_o$ pulse is lost and the control sequence hangs-up i.e., if $\alpha = \beta = 0$, an error exists.

In many cases the conditional logic is designed such that $\alpha = \overline{\beta}$ and this error situation is averted.

The sequence stage may also be required to delay the continuation of control contingent upon an asynchronous wait condition. Since the arrival time of this signal is unknown, the circuit configuration of Figure 3.2.1 is not applicable. Figure A.2.2 and Figure A.2.3 illustrates methods of implementing this wait condition.



α, β ARE BRANCHING CONDITIONS

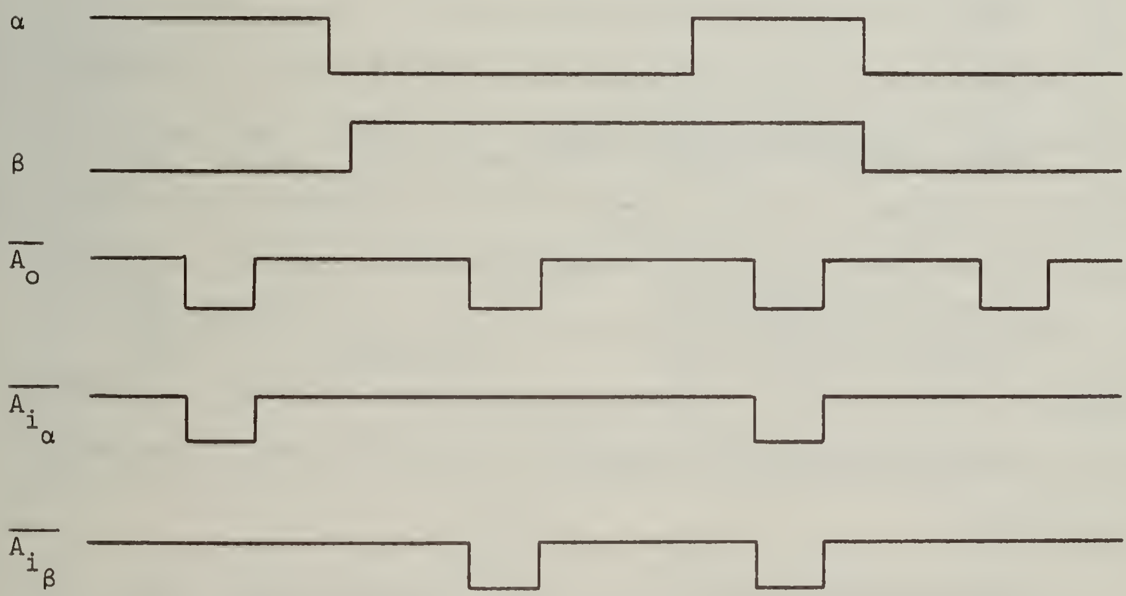


Figure A.2.1 Two Way Branch Sequence Stage Logic

In Figure A.2.2 the wait signal is used to delay the action of control point after it has been initiated. That is, $\overline{DO}$ will be inhibited from going to "0" until WAIT goes to "0". When the wait condition is satisfied, $\overline{DO}$ provides a means for keeping EN at "1" until the control point completes its normal operation. This same delay action to the control point operation could also be a function of a maintenance halt, MH, used for check-out and diagnostic procedures.

In Figure A.2.3 the wait signal is used to delay the propagation of the $\overline{A}_o'$ signal. The $\overline{DO}$ signal is then a function of both the timing stage and the wait condition. Here the $\overline{DO}$ line will drop to "0" and remain there (even though the timing stage is done) until WAIT goes to "0". The $\overline{A}_o'$ signal will then terminate the operation of the control point and initiate the next stage logic.

This same wait logic configuration may also be used to interlock two or more parallel, independent control chains. The design requirement here is to make the $\overline{A}_o$ signal to the next stage wait until all the parallel tasks are complete. Figure A.2.4 illustrates an example of this interlocked control chains. The $\overline{A}_o$ line of last control point of each chain is delay until a reply from all the chains is received. It is assumed that when one control chain is activated, then so is the other, i.e., that eventually both replies will be generated.

Another application of the wait logic, similar to that shown in Figure A.2.3, is where a control point is used to call some routine and will not advance control until the called routine has replied. For apparent reasons, this configuration has been named "calling control point" and is

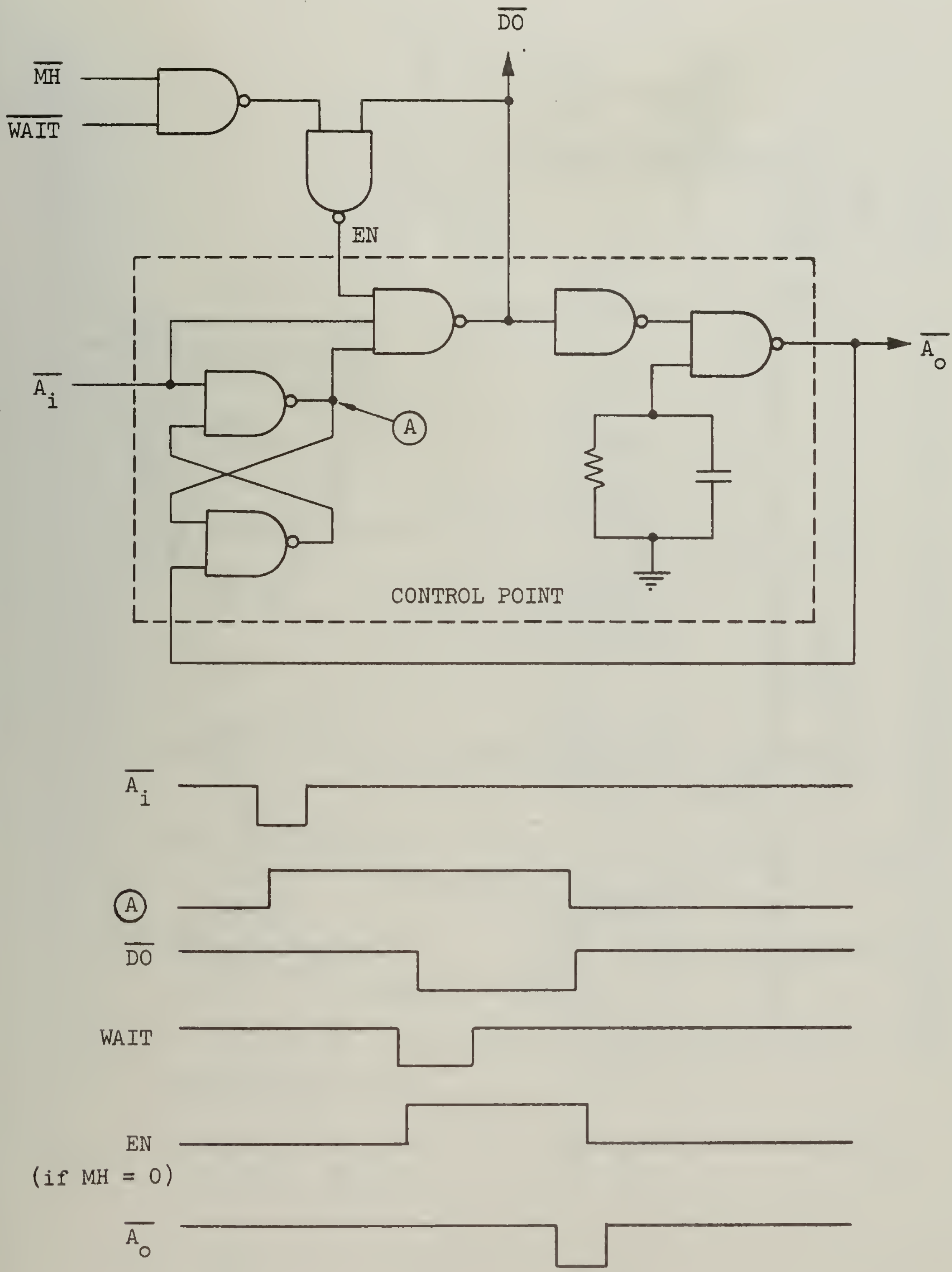
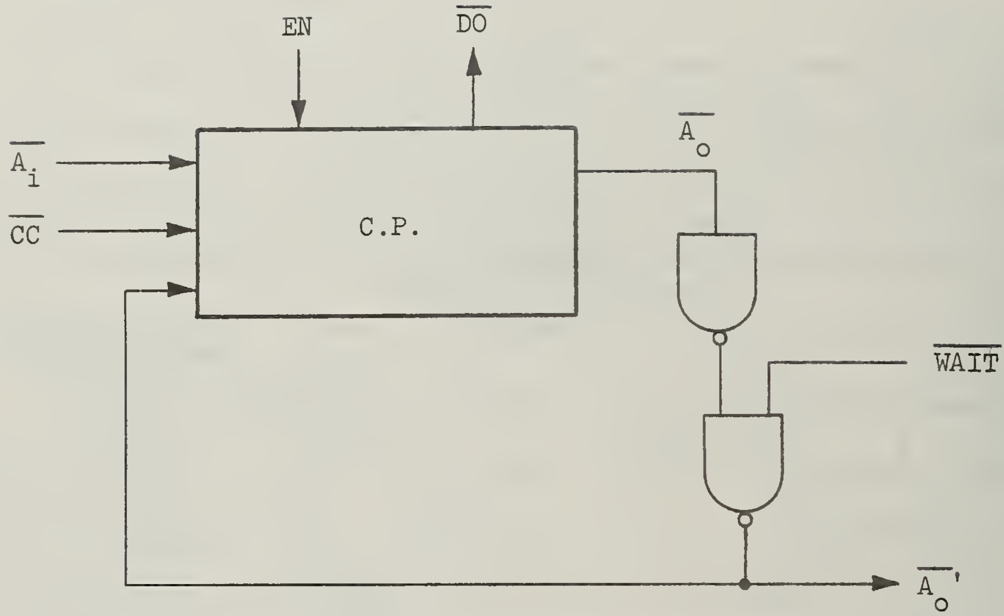


Figure A.2.2 WAIT Condition Using EN Gate



ASSUME THAT EN = $\overline{CC}$ = 1

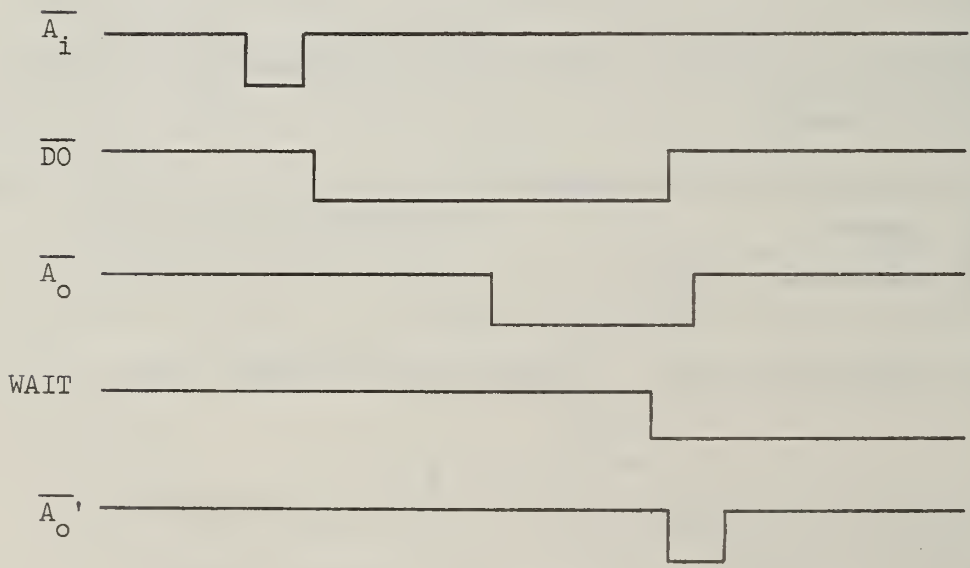
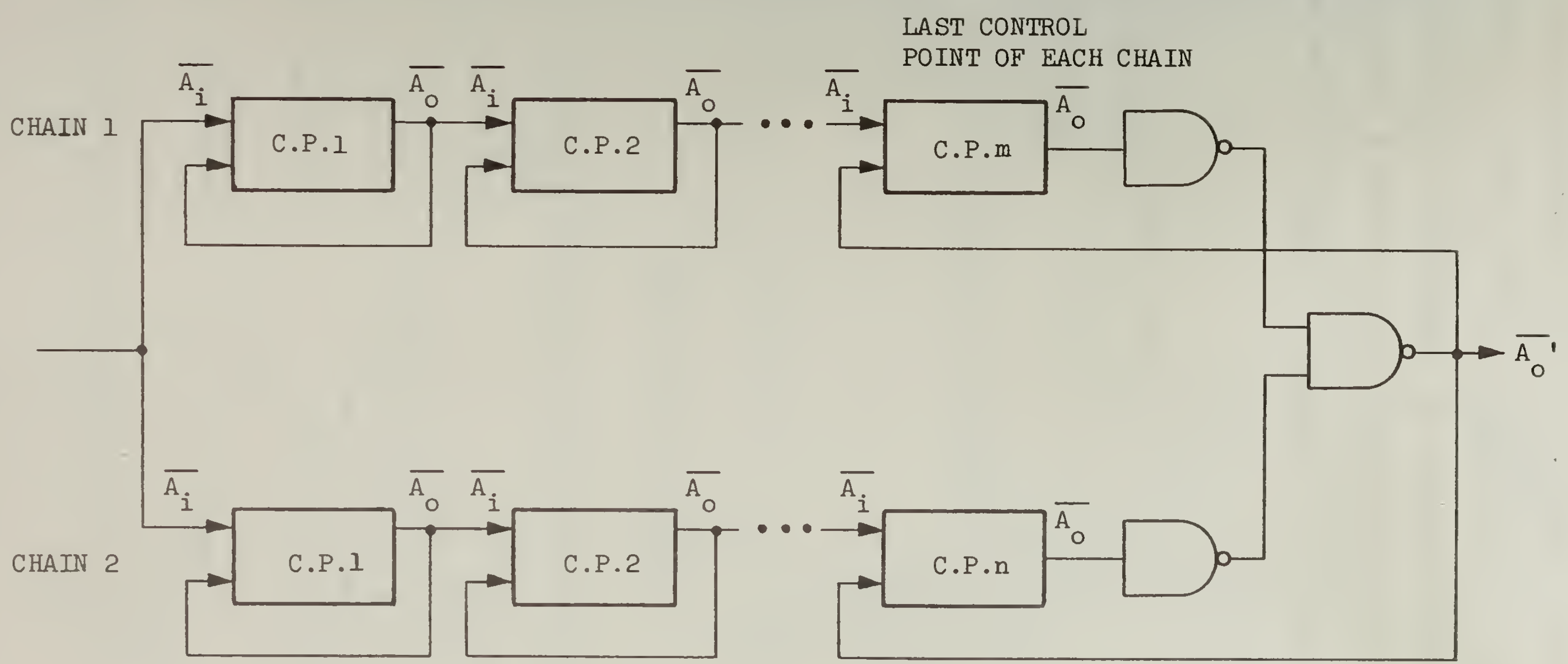


Figure A.2.3 Wait Condition Using $\overline{A_o}$ Line

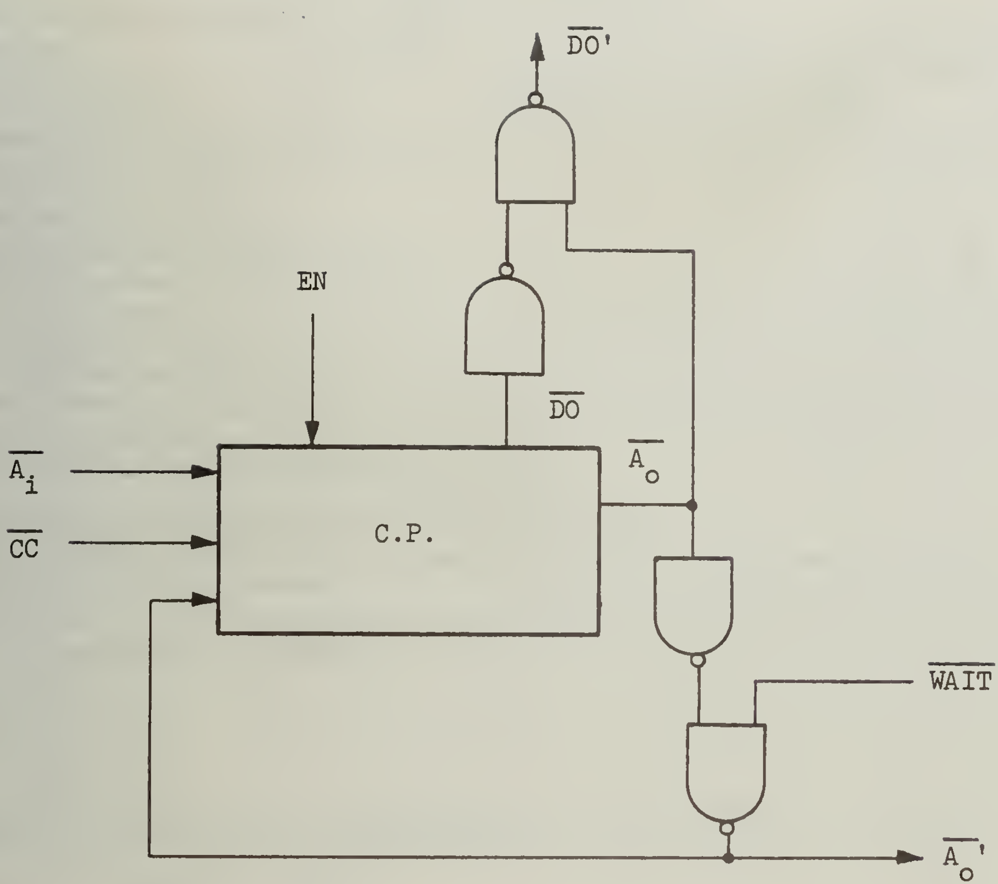


NOTE THAT EACH CHAIN NOT NECESSARILY THE SAME LENGTH

i.e. $m \neq n$

Figure A.2.4 Interlocking Two Parallel Control Chains

shown in Figure A.2.5. The $\overline{DO}'$ line will go to "0" as the control point is initiated (i.e., $\overline{DO}$ goes to "0" and $\overline{A}_o$ is at "1"). After a specific delay time, due to the timing stage, $\overline{A}_o$ goes to "0" causing $\overline{DO}'$ to return to "1". This action of $\overline{DO}'$ may have been used to prime and initiate a control point in a routine. When this routine has completed its task, it replies and WAIT goes to "0".



ASSUME THAT EN = $\overline{CC}$ = 1

Figure A.2.5 "Calling Control Point" Circuit Configuration

orm AEC–427
(6/68)
AECM 3201

U. S. ATOMIC ENERGY COMMISSION
UNIVERSITY–TYPE CONTRACTOR'S RECOMMENDATION FOR DISPOSITION OF SCIENTIFIC AND TECHNICAL DOCUMENT

(See Instructions on Reverse Side)

. AEC REPORT NO.	2. TITLE
COO-2118-0041	ILLIAC III COMPUTER SYSTEM MANUAL: ARITHEMTIC UNITS--VOLUME 2

3. TYPE OF DOCUMENT (Check one):

- [X] a. Scientific end technical report
- [ ] b. Conference paper not to be published in e journel:
  - Title of conference ____
  - Date of conference ____
  - Exact location of conference ____
  - Sponsoring organization ____
- [ ] c. Other (Specify) ____

4. RECOMMENDED ANNOUNCEMENT AND DISTRIBUTION (Check one):

- [X] a. AEC's normel announcement and distribution procedures may be followed.
- [ ] b. Make aveileble only within AEC and to AEC contractors and other U.S. Government agencies end their contractors.
- [ ] c. Make no announcement or distrubution.

5. REASON FOR RECOMMENDED RESTRICTIONS:

6. SUBMITTED BY: NAME AND POSITION (Pleese print or type)

Lakshmi N. Goyal
Research Assistant

Organization

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

Signature	Date
Lakshmi N. Goyal	January 1973

FOR AEC USE ONLY

7. AEC CONTRACT ADMINISTRATOR'S COMMENTS, IF ANY, ON ABOVE ANNOUNCEMENT AND DISTRIBUTION RECOMMENDATION:

--

8. PATENT CLEARANCE:

- [ ] a. AEC patent clearance has been granted by responsible AEC patent group.
- [ ] b. Report hes been sent to responsible AEC patent group for clearance.
- [ ] c. Patent clearance not required.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-72-551	2.	3. Recipient's Accession No.
4. Title and Subtitle ILLIAC III COMPUTER SYSTEM MANUAL: ARITHMETIC UNITS--VOLUME 2			5. Report Date November 1972 6.
7. Author(s) Lakshmi N. Goyal			8. Performing Organization Rept. No. COO-2118-0041
9. Performing Organization Name and Address Department of Computer Science University of Illinois Urbana, Illinois 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No. US AEC AT(11-1)2118
12. Sponsoring Organization Name and Address US AEC Chicago Operations Office 9800 South Cass Avenue Argonne, Illinois			13. Type of Report & Period Covered systems manual 14.

15. Supplementary Notes

16. Abstracts

This report is the second volume of Illiac III computer system manual on arithmetic units. The first volume described the various arithmetic orders to be executed in the arithmetic unit and the internal statis description or in other words, the processing hardware of the Arithmetic Unit. This volume describes the control logic hardware and gives its operational description.

This report assumes that the reader is familiar with previously published reports and documents regarding the arithmetic.

17. Key Words and Document Analysis. 17a. Descriptors

strategy of control design
control point flow charts
control logic
processing hardware interface
point design
arithmetic
addition, subtraction
multiplication, division, number conversion
signed-digit arithmetic

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
availability unlimited	UNCLASSIFIED	318
	20. Security Class (This Page) UNCLASSIFIED	22. Price

FORM NTIS-35 (10-70) USCOMM-DC 40329-P71